OMNIC™

Macros\Pro User's Guide

Version 7.3

For technical assistance, please contact:
Technical Support
Thermo Electron Corporation
5225 Verona Road
Madison WI 53711-4495
U.S.A.

Telephone: 1 800 642 6538 (U.S.A.) or +1 608 273 5015 (worldwide)
Fax: +1 608 273 5045 (worldwide)
E-mail: techsupport.analyze@thermo.com

269-043706

Contents

Introduction

Macros\Pro™ is an advanced macros development package for OMNIC® that is easy to use but powerful enough to let you develop macros that will perform complex operations. The components of Macros\Pro include:

- OMNIC commands and parameters. The operations a macro performs are specified with commands and parameters. Any operation available in OMNIC can be performed using commands and parameters.

- OmTalk routines that manage the communications between OMNIC and Visual Basic®. These routines allow you to execute OMNIC commands, set and read OMNIC parameter values, and perform a number of other interactions between OMNIC and Visual Basic.

- The Visual Basic programming software. With Visual Basic you can create sophisticated user interfaces and program your macros to perform a wide variety of operations.

On-line help for Macros\Pro

To use the on-line help system for Macros\Pro, click the Windows® Start menu, point to Programs, and then point to the OMNIC folder. This opens a menu of the components of OMNIC you have installed. Click OMNIC Macros\Pro Help to open the Macros\Pro help system.

About Macros\Pro

Macros\Pro version 6.1a and greater is designed to work with Visual Basic 6.0 and Visual Basic.NET. Here are some things you should keep in mind when using Visual Basic.

- If you are using Visual basic version 6.0 with Macros\Pro version 6.1a and greater, the OmTalk files OMTALK.FRM and OMTALK.BAS support compilation for 32-bit applications. This occurs automatically; you do not have to do anything other than add these two files to your project.

- If you are using Visual Basic.NET with Macros\Pro version 6.1a and greater, the OmTalk file OMTALK32.VB supports compilation for 32-bit applications. This occurs automatically; you do not have to do anything other than add this file to your project.

Note You can use Visual Basic 6.0 with OmTalk.NET. To do so, use the OMTALK.EXE and OMTALK32.BAS files and follow the instructions for OmTalk.NET. ▲

- Example macros that you can look at using Visual Basic are included with the Macros\Pro software. These examples are in a folder within the OMNIC\PROMACS\EXAMPLES directory. (If you have Visual Basic 6.0, you can use the Visual Basic 5.0 examples.)

Note If you are using Visual Basic 3.0 or earlier, you must use the 16-bit versions of the OmTalk files, OMTALK16.FRM and OMTALK16.BAS. When creating a new Visual Basic 3.0 project, be sure to add the OMTALK16 files to your project, not the OMTALK files. If you are working with an existing application, you must remove the OMTALK files, and then add the OMTALK16 files to your project. ▲

Note Visual Basic.net is used to refer to Visual Basic under various names, including Visual Basic.net, Visual Basic.net 2003, Visual Basic 2005, or other future Microsoft product names. It is currently available only as a component of Visual Studio. ▲

- If you are using a 32-bit version of Visual Basic, you must add the OmnicCom 1.0 Type Library to your project before attempting to make an .exe file. If you do not, and you are using Visual Basic 6.0, you will get an error message. (For more information, see “Referencing type libraries” in this chapter.)

- If you are using a Visual Basic.NET, you must add the OmTalk.NET 1.0 Type Library to your project before attempting to make an .exe file. If you do not, you will get a build error. (For more information, see “Referencing type libraries” in this chapter.)

About this manual

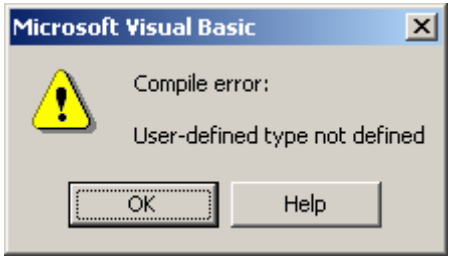
This manual is divided into the following chapters:

- **Creating Pro Macros With Visual Basic**
  Provides information about creating Pro macros using Visual Basic and OmTalk.
- **Using a Pro Macro in OMNIC**
  Describes how to run your Pro macros from OMNIC, using menu commands or buttons.
- **Using the OmTalk Visual Basic Routines**
  Explains how to use the OmTalk routines to handle communications between Visual Basic and OmTalk. A detailed description of each OmTalk routine is provided.
- **The OMNIC DDE Application**
  Provides information on the stand-alone application for sending commands to OMNIC and for setting and getting the values of OMNIC parameters via dynamic data exchange (DDE).
- **OMNIC Commands and Parameters**
  Provides information about using the OMNIC commands and parameters.
- **Macros\Pro Examples**
  Provides brief descriptions of the content and purpose of each of the example macros provided with your Macros\Pro software.
- **Using OMNIC Commands and Parameters with Other Applications**
  Provides information on using the OMNIC commands and parameters with applications other than Visual Basic.

Note You will need to know how to program in Visual Basic before you can create Pro macros. For information about programming in Visual basic, see the Visual Basic documentation and tutorial. ▲

Referencing type libraries

Pro macros that you create or run with OMNIC 5.1 or higher must reference the OmnicCom 1.0 Type Library (and the OmTalk.NET Type Library if you are using Visual Basic.NET). If you do not add the OmnicCom 1.0 Type Library to a Visual Basic 6.0 project, you will get the following error message:



If you are using Visual Basic.NET, and you do not add the OmTalk.NET Type Library to your project, you will get the following build error:

“Type 'OmTalk.OmTalkClass' is not defined.”

The procedures that follow explain how to reference the appropriate type libraries in a Pro macro (or to update an existing Pro macro to run with OMNIC 5.1 or higher) when you are using Macros\Pro version 6.1 or greater.

Referencing type libraries with Visual Basic 6.0

To reference the OmnicCom 1.0 Type Library when using Visual Basic 6.0, follow these steps:

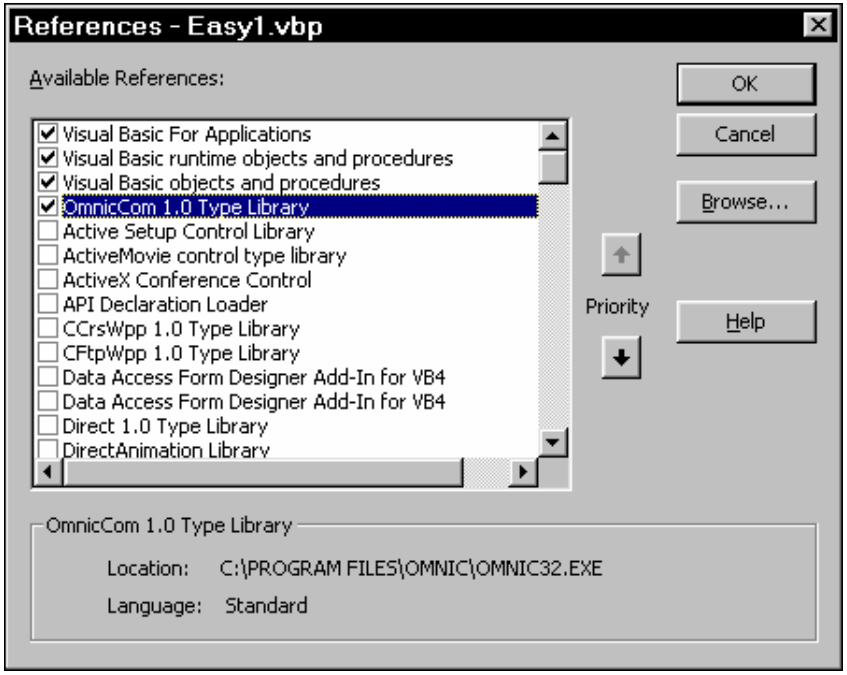
1. **Start Visual Basic.**

2. **Create or open the macro you want to update.**

3. **Choose References from the Project menu.**

   The References dialog box appears on the display.

4. **Turn on the check box for OmnicCom 1.0 Type Library in the Available References list.**



5. **Click OK to close the References dialog box.**

6. **Use the Save Project command in the Visual Basic File menu to save the macro.**

7. **Use the Make command in the Visual Basic File menu to create an executable (.EXE) file for your macro.**

Referencing type libraries with Visual Basic.NET

To reference the OmnicCom 1.0 Type Library and OmTalk.NET library when using Visual Basic.NET, follow these steps:

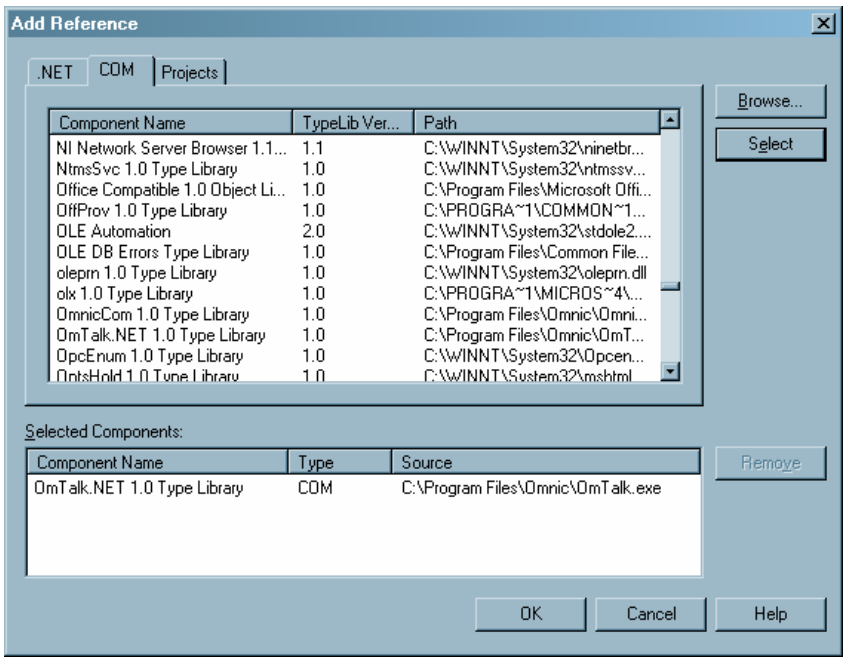
1. **Start Visual Basic.NET.**

2. **Create or open the macro you want to update.**

3. **Choose Add Reference from the Project menu.**

   The Add Reference dialog box appears on the display.

4. **Click the COM tab, from the list of available references, select OmnicCom 1.0 Type Library and click Select, and then select OmTalk.NET and click Select.**



5. **Click OK to close the References dialog box.**

6. **Use the Save All command in the File menu to save the macro.**

7. **Use the Build Solution command in the Build menu to create an executable (.EXE) file for your macro.**

Creating Macros With Visual Basic

Once you are familiar with creating Visual Basic applications, you can use Visual Basic to create Pro macros. If you have not programmed with Visual Basic before, you should become familiar with it before continuing. The on-line Help system included with the Visual Basic software provides a good overview of the process for creating Visual Basic applications.

The OmTalk routines can be used within your Visual Basic application to pass information such as command instructions, parameter settings and spectral data between Visual Basic and OMNIC. These routines handle the communication between Visual Basic and OMNIC so that you don't have to program the interactions between the two applications. For a more complete description of the OmTalk routines, see "Using the OmTalk Visual Basic Routines" chapter of this manual.

Pro macros are created using the same basic steps you would use for creating any Visual Basic application. The following procedure leads you through creating the interface, incorporating the OmTalk routines, and writing the code for the example macro EASY1.VBP.

To view the completed EASY1.VBP form and associated code, open the EASY1.VBP project in Visual Basic. If you have already installed the Macros\Pro software, EASY1.VBP is located in the OMNIC\PROMACS\EXAMPLES\VB#\EASY1 directory on your hard disk, where VB# corresponds to the version number of Visual Basic you are using.

Note If you are using Visual Basic 3.0, the example file is named EASY1.MAK. ▲

Creating pro macros with Visual Basic 6.0

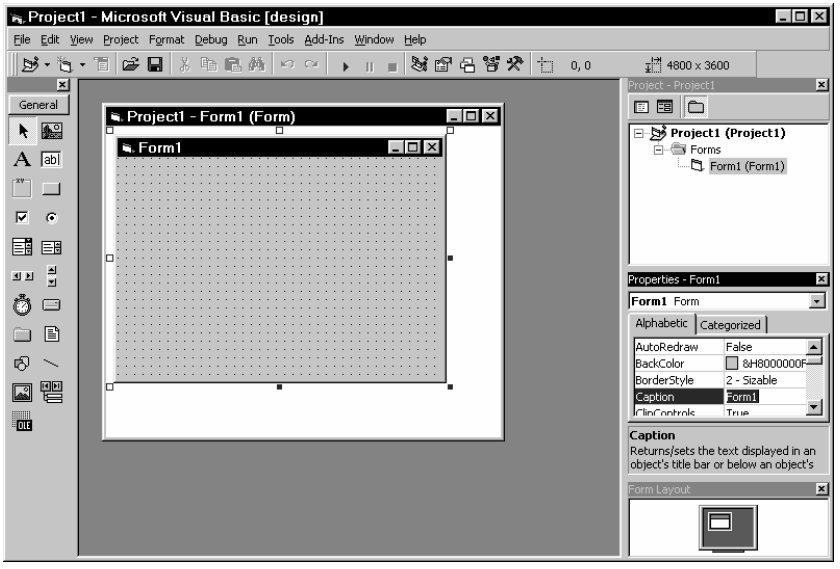
This example shows how to create a Pro macro with Visual Basic 6.0 and OmTalk. If you're using an earlier version of Visual Basic, the same directions apply in general. If you're using Visual Basic.NET with OmTalk.NET, an applicable example follows this example.

1. **Start Visual Basic.**

   Use the Start button to go to the Visual Basic folder, and then click the Visual Basic program name. You can also use Explorer to find and click the Visual Basic program file.

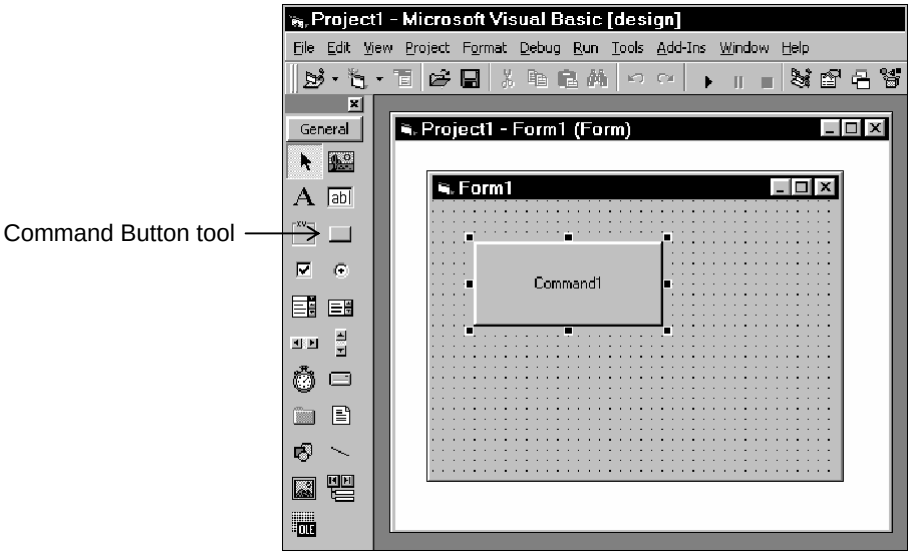
2. **Create a new project.**

   If the New Project window (see illustration below) does not appear, display it by choosing New Project from the File menu. Then click the Standard EXE icon and click Open. Visual Basic creates a new project with a Form window as shown below. You use the Form window to create the controls you want your users to see.



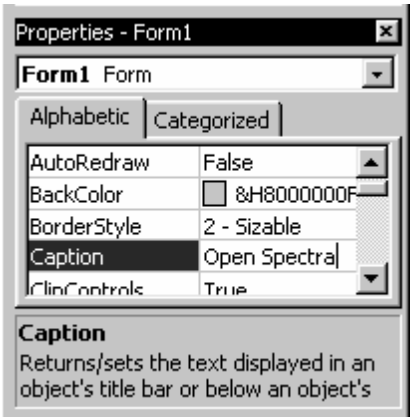
3. **Select the Command Button tool and create the first of two buttons in this example macro.**

   Click the Command Button tool once. Then go to the Form window and drag with the mouse to create the button.



4. **Define the button name or caption.**

   Display the button Properties window if it is not already shown. (Right-click the button and choose Properties from the drop-down menu.) If you display the Properties window immediately after creating a button, the Caption field will be selected automatically.



   With the current caption highlighted, type the name "Open Spectra." Click the Close button to close the Properties window. The new name appears on the button. You can use the Properties window to quickly change a button caption or any other button property at any time.

5. **Repeat the previous steps to create another button with the caption "Exit".**

6. **Add the OMTALK.FRM and OMTALK.BAS files to your project.**

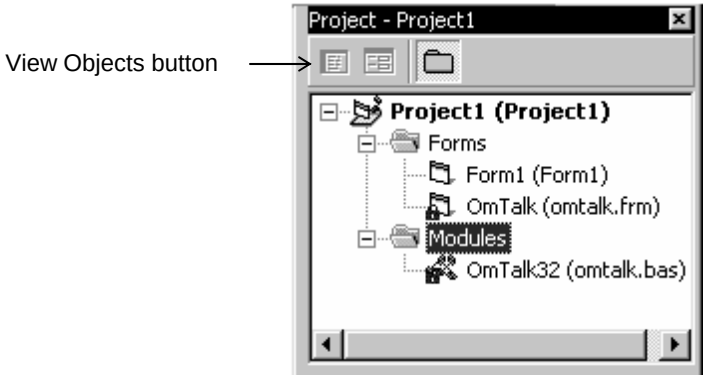
   These files must be included in the project in order to use the OmTalk commands. If you are using Visual Basic 3, you must use the 16-bit versions of these files (OMTALK16.FRM and OMTALK16.BAS).

   To add the OMTALK.FRM file to your project, choose Add Form from the Project menu. In the Add Form window, click the Existing tab and open the file OMTALK.FRM (or OMTALK16.FRM) from the PROMACS directory of your OMNIC data directory. Click OK when the following message appears.

   "This file was saved in a previous version of Visual Basic. When saved, it will be saved in the Visual Basic 5.0 format."

   To add the OMTALK.BAS file to your project, choose Add Module from the Project menu. In the Add Module window, click the Existing tab and open the file OMTALK.BAS (or OMTALK16.BAS) from the PROMACS directory of your OMNIC data directory.

   Verify that the files are loaded by checking for the OmTalk files in the Project window as shown below.

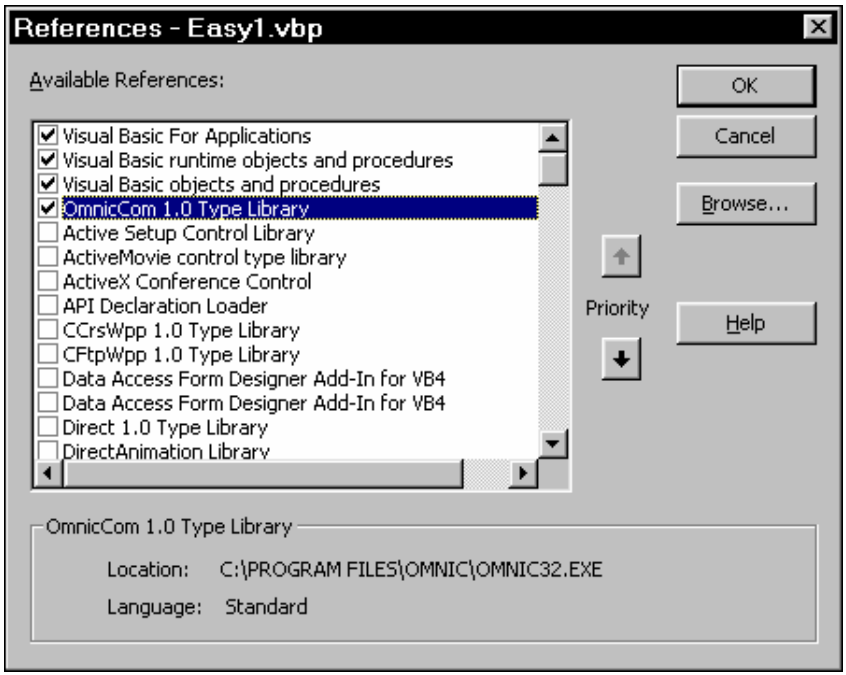


7. **Add the OmnicCom 1.0 Type Library reference to your project.**

   From the Visual Basic Project menu, choose References.

   The References dialog box appears on the display.

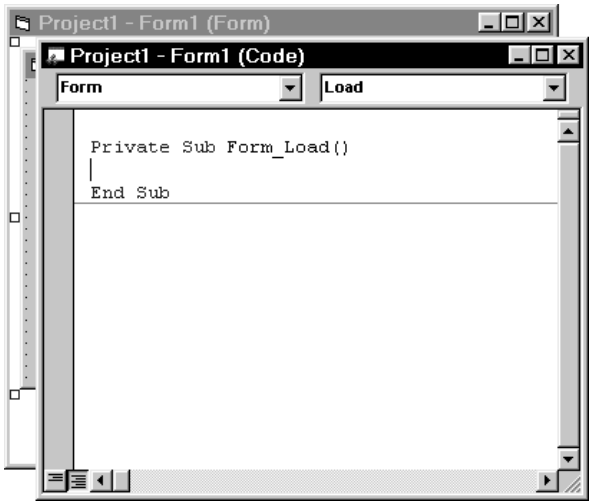
   Scroll the list of available references until the OmnicCom 1.0 Type Library check box is displayed and then turn on the check box as shown in the following illustration.



   Click OK to close the dialog box.

8. Write the code for the form.

The form and each object on the form can have associated code. To open the Code window for the form, double-click a blank area in the Form window (don't click one of your buttons).



Visual Basic begins some of the coding for you by entering the start and end of the sections. Complete the code as shown below. Lines beginning with a single apostrophe (') are comments and may be omitted.

```
Private Sub Form_Load()
'Load OMTALK services and maximize OMNIC application
'window.
  Load OMTALK
  ExecuteOMNIC "MaximizeWindow"
End Sub
```

Note Refer to "Using the OmTalk Visual Basic Routines" chapter for information about specific OmTalk routines. Refer to the Macros\Pro on-line help for a complete list of the OMNIC commands and parameters and for information on the command and parameter syntax. ▲

9. Write the code for the OpenSpectra button.

Double-click the Open Spectra button on the form to display the Code window for the button. If the form is hidden, use the View Objects button on the Project Window toolbar to display it.

Enter the code for the button as shown below.

```
Private Sub OpenSpectra_Click()
Dim CmdStr As String
Dim lvOMNICDataDir As String
Dim lvOMNICName As String
Dim lvEZOMNICName As String
Const lcSPECTRA = "spectra\"
Const lcSPA = ".spa"

'This is a very simple example using the OMTALK DDE
subroutines.
'Size the OMNIC window and move it to the upper left
corner of the screen.
'Open a new spectral window.
    ExecuteOMNIC "SizeWindow 400 480"
    ExecuteOMNIC "MoveWindow 0 0"
    ExecuteOMNIC "NewWindow"
'Set the number of panes to two.
    SetOMNIC "Display Panes", 2
'Set the display to the stack mode.
    SetOMNIC "Display Mode", "Stackmode"
'Find out where OMNIC Data Dir is.
Call FindOMNICData(lvOMNICDataDir, lvOMNICName,
lvEZOMNICName)
lvOMNICDataDir$ = lvOMNICDataDir$ & "\"
'Open two of the standard example spectra and
'place them in the new spectral window.
    CmdStr$ = """ & lvOMNICDataDir$ & lcSPECTRA &
"fsd" & lcSPA & """
    ExecuteOMNIC "Import " & CmdStr$
    CmdStr$ = """ & lvOMNICDataDir$ & lcSPECTRA &
"fsd" & lcSPA & """
    ExecuteOMNIC "Import " & CmdStr$
'Bring Form1 to the front.
    Form1.Show

End Sub
```

Note Lines that begin with a single apostrophe (') are comments and may be omitted from your code. ▲

10. Write the code for the Exit button.

Double-click the Exit button on the form to display the code window for that button. If the form is hidden, use the View Objects button on the Project Window toolbar to display it.

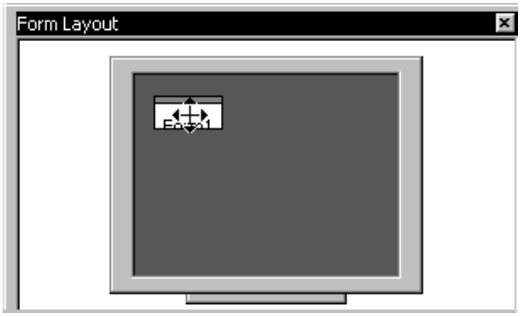
```
Private Sub Exit_Click ()

'Exit Visual Basic Program
  End

End Sub
```

You can learn more about the Macros\Pro commands and parameters used in this example by referring to the command and parameter descriptions in the Macros\Pro on-line help.

11. Set the location of the form on the screen.

From the View menu, choose Form Layout Window.

The Form Layout window appears with a box that represents your macro.



Drag the box in the Form Layout Window to set the location of the form on the screen when your Pro macro is running.

12. **Save the project.**

    Use the Save Project command in the File menu to save the project. We recommend using the same name for all of the files, forms and modules associated with a project.

13. **Test the project.**

    You should test a new project before saving it as an executable (.EXE) file. To test the project, choose Start from the Visual Basic Run menu. The project will open and run as if you had already made the executable. If the project works as you intended and there are no error messages, you are ready to create the executable version of your project. Choose End from the Run menu to stop the test.

14. **Create an executable file.**

    Use the Make command in the File menu to create an executable (.EXE) file for your application.

15. Run the Pro macro .EXE file.

You can use any of the following methods to run a Pro macro.

- To run a Pro macro from OMNIC, you must first add the macro to an OMNIC menu or the OMNIC toolbar. For more information, see the chapter in this manual titled "Using a Pro Macro in OMNIC."
- If you purchased the OMNIC Macros\Basic software, you can insert a Pro macro into a Basic macro using the Macro command in the Macros\Basic Insert menu. For more information, see the chapter in this manual titled "Using a Pro Macro in OMNIC."
- To run a Pro macro from Microsoft® Windows, open the Windows Start menu, choose the Run command and specify the .EXE file to be run. You can also run an executable program from Windows Explorer or My Computer by double-clicking the icon for the executable file. For more information on running programs from Windows, see your Windows documentation.

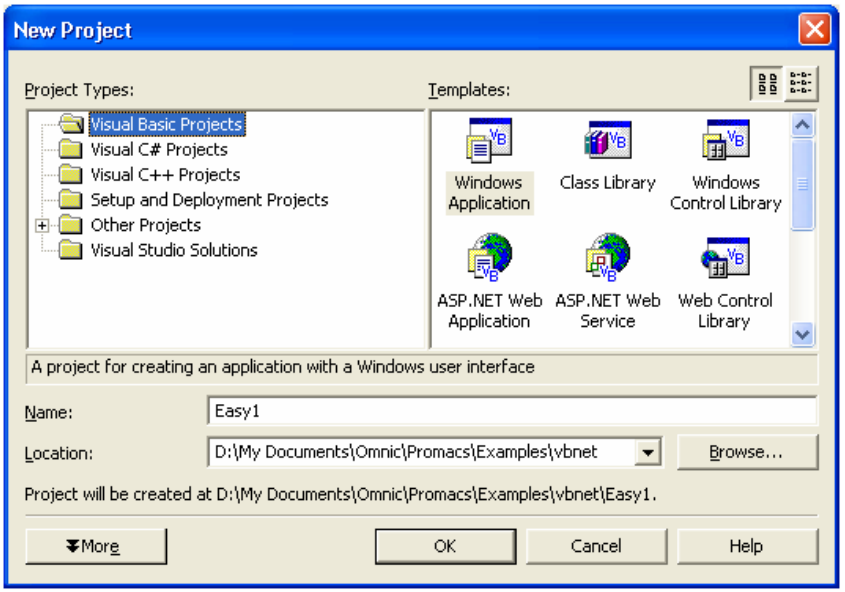
Creating pro macros with Visual Basic.NET

This example shows how to create a Pro macro with Visual Basic.NET and OmTalk.NET. If you're using Visual Basic 6.0 (or earlier) refer to the previous example.

1. **Start Visual Basic.**

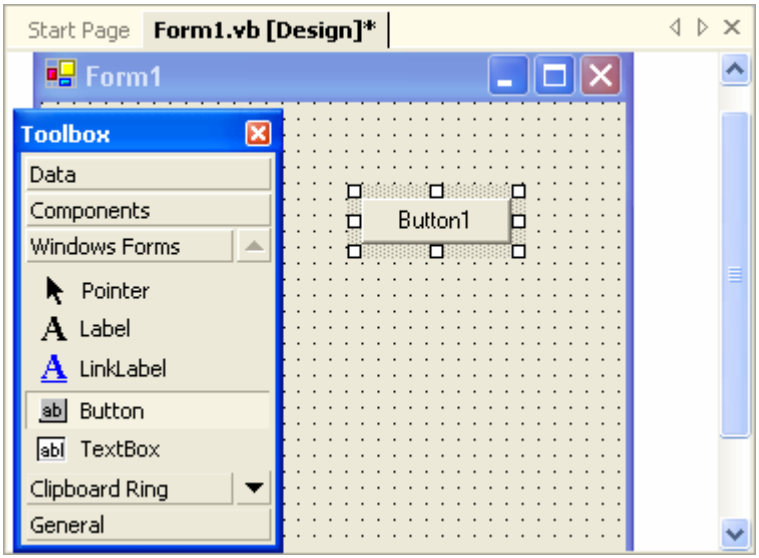
2. **Create a new project.**

   Choose New Project form the File menu. When the New Project dialog box opens, select Visual Basic Projects, and click Windows Application. Enter a name for the project in the Name box and choose a location for the project from the Location drop-down list box.



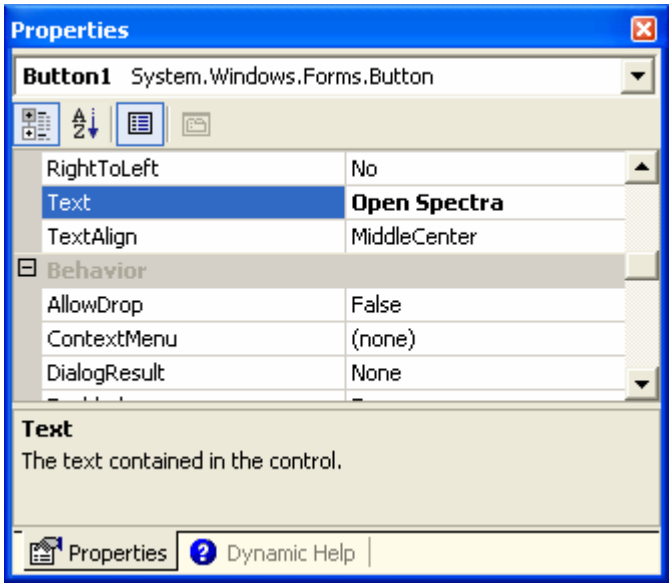
3. **Open the Toolbox, select the Button tool, and create the first of two buttons in this example macro.**

   Click the Button tool. Then go to the Form window and drag with the mouse to create the button.



4. **Define the button name or caption.**

   Display the button Properties window if it is not already shown. (Right-click the button and choose Properties from the drop-down menu.) If you display the Properties window immediately after creating a button, the Caption field will be selected automatically.



   With the current caption highlighted, type the name "Open Spectra." When you close the Properties window, the new name appears on the button. You can use the Properties window to quickly change a button caption or any other button property at any time.

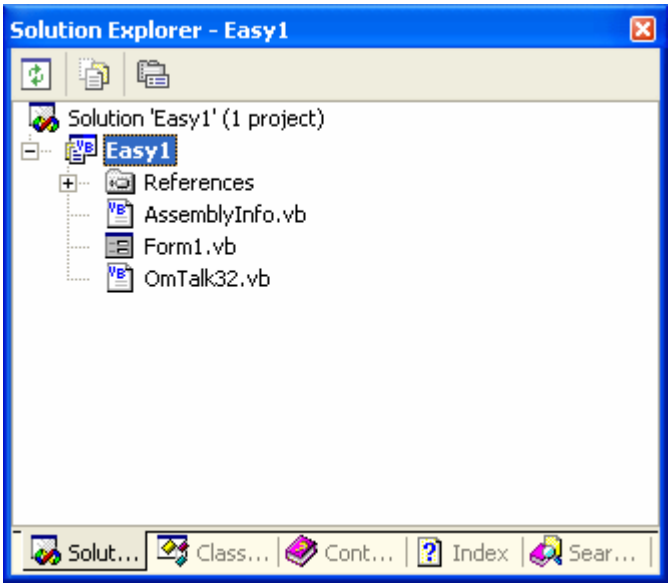
5. **Repeat the steps 3 and 4 to create another button with the caption "Exit".**

6. **Add the OMTALK32.VB file to your project.**

   These files must be included in the project in order to use the OmTalk commands.

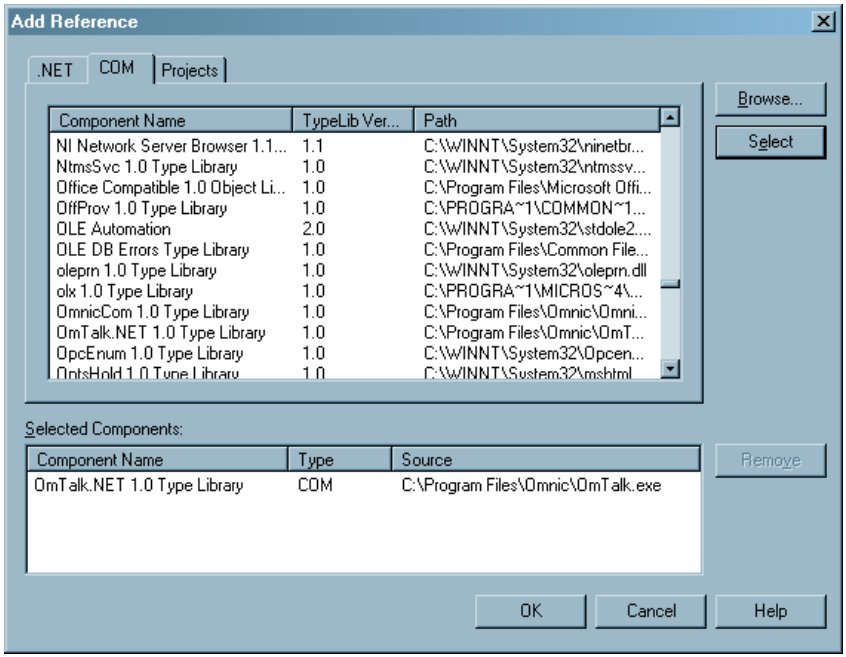
   To add the OMTALK32.VB file to your project, choose Add Existing Item from the Project menu. In the Add Existing Item dialog box open the OMTALK32.VB file from the PROMACS directory of your OMNIC data directory.

   Verify that the files are loaded by checking for the OmTalk files in the Project window as shown below.



7. Add references to your project.

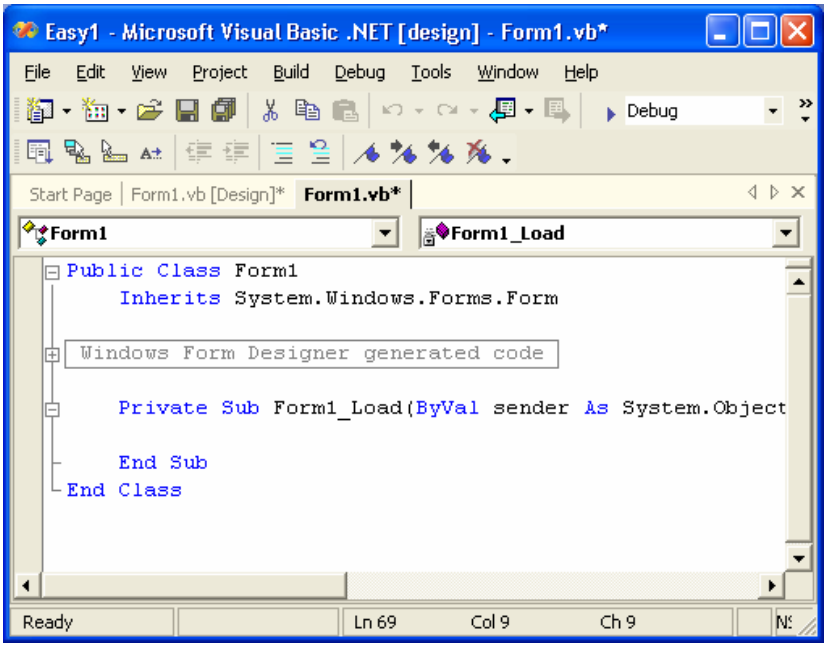
Choose Add Reference from the Visual Basic Project menu, and then click the COM tab in the Add Reference dialog box. Select OmTalk.NET 1.0 Type Library from the list of available references and click Select.



Click OK to close the dialog box.

8. Write the code for the form.

The form and each object on the form can have associated code. To view the code for the form, double-click a blank area in the form window.



Visual Basic.NET begins some of the coding for you by entering the start and end of the sections. Complete the code as shown below.

```
Private Sub Form_Load ()
   LoadOmTalk()
   ExecuteOMNIC ("MaximizeWindow")
End Sub
```

Note Refer to the "Using the OmTalk.NET Visual Basic Routines" chapter for information about the OmTalk routines. Refer to the Macros\Pro on-line help for information about OMNIC commands and parameters. ▲

9. Write the code for the OpenSpectra button.

Double-click the Open Spectra button on the form to display the code for the button. If the form is hidden, choose Designer from the View menu to display it.

Enter the code for the button as shown below.

```
Private Sub Button1_Click(ByVal sender As
↙System.Object, ByVal e As System.EventArgs)
↙Handles Button1.Click
   Dim CmdStr As String
   Dim fvOMNICDataDir As String
   Dim fvOMNICName As String
   Dim fvEZOMNICName As String
   Const lcSPECTRA = "spectra\"
   Const lcSPA = ".spa"

   'This is a very simple example using the OMTALK
   ↙DDE subroutines.
   'Size the OMNIC window and move it to the upper
   ↙left corner of the screen.
   'Open a new spectral window.
   ExecuteOMNIC("SizeWindow 400 480")
   ExecuteOMNIC("MoveWindow 0 0")
   ExecuteOMNIC("NewWindow")
   'Set the number of panes to two.
   SetOMNIC("Display Panes", 2)
   'Set the display to the stack mode.
   SetOMNIC("Display Mode", "Stackmode")
   'Find out where OMNIC Data Dir is.
   Call FindOMNICData(fvOMNICDataDir, fvOMNICName,
   ↙fvEZOMNICName)
   fvOMNICDataDir$ = fvOMNICDataDir$ & "\"
   'Open two of the standard example spectra and
   'place them in the new spectral window.
   CmdStr$ = """ & fvOMNICDataDir$ & lcSPECTRA &
   ↙"fsd" & lcSPA & """
   ExecuteOMNIC("Import " & CmdStr$)
   CmdStr$ = """ & fvOMNICDataDir$ & lcSPECTRA &
   ↙"fsd" & lcSPA & """
   ExecuteOMNIC("Import " & CmdStr$)
End Sub
```

10. Write the code for the Exit button.

Double-click the Exit button on the form to display the code window for that button. If the form is hidden, choose Designer from the View menu to display it.

```
Private Sub Exit_Click()
   UnloadOmTalk()
   'Exit Visual Basic Program
   End

End Sub
```

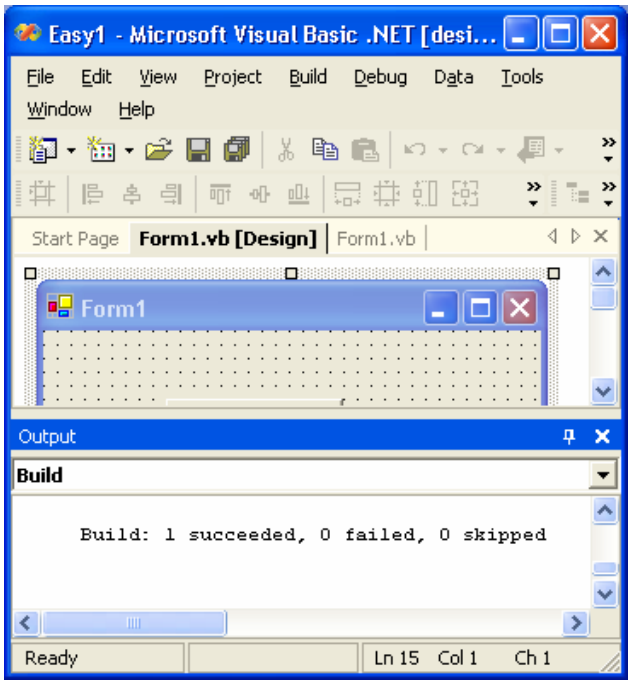
Note For information about the OMNIC commands and parameters, see the Macros\Pro on-line help. ▲

11. Save the project.

Choose Save All from the File menu to save the project. We recommend using the same name for all of the files, forms and modules associated with a project.

12. Build the project.

Choose “Build Easy 1” from the Build menu. The Output dialog box will display a message telling you if the build succeeded.



13. Test the project.

You should test a new project before saving it as an executable (.EXE) file. To test the project, choose Start from the Debug menu. The project will open and run as if you had already made the executable. If the project works as you intended and there are no error messages, you are ready to create the executable version of your project.

14. Create an executable file.

Choose Configuration Manager from the Build menu, and set Active Solution Configuration to Release. Next, use the Rebuild command in the Build menu to create the executable.

15. Run the Pro macro .EXE file.

You can use any of the following methods to run a Pro macro.

- To run a Pro macro from OMNIC, you must first add the macro to an OMNIC menu or the OMNIC toolbar. For more information, see the chapter in this manual titled "Using a Pro Macro in OMNIC."
- If you purchased the OMNIC Macros\Basic software, you can insert a Pro macro into a Basic macro using the Macro command in the Macros\Basic Insert menu. For more information, see the chapter in this manual titled "Using a Pro Macro in OMNIC."
- To run a Pro macro from Microsoft® Windows, open the Windows Start menu, choose the Run command and specify the .EXE file to be run. You can also run an executable program from Windows Explorer or My Computer by double-clicking the icon for the executable file. For more information on running programs from Windows, see your Windows documentation.

Using a Pro Macro in OMNIC

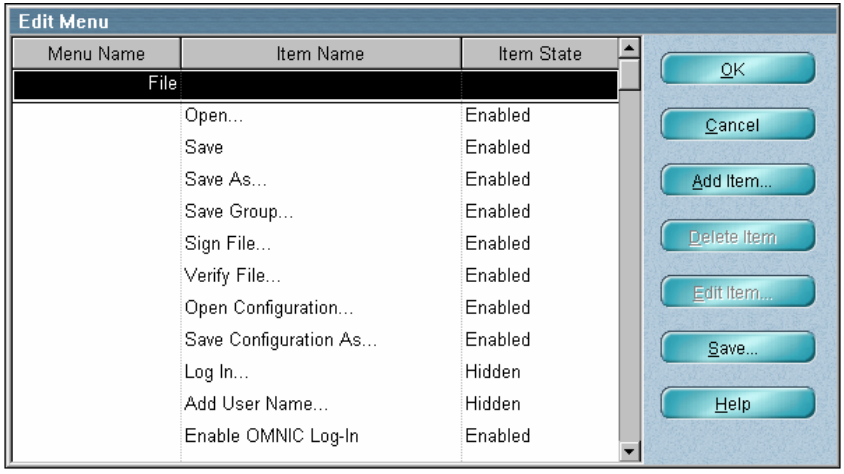
There are three ways to execute Pro macros directly from OMNIC: you can add a macro to an OMNIC menu or the OMNIC toolbar, or you can call the macro from the Macros\Basic program. This chapter provides step-by-step instructions for each method.

Adding a macro to an OMNIC menu

The Edit Menu command in the OMNIC Edit menu allows you to customize the menus by adding macro commands. To assign a macro to an OMNIC menu, follow these steps:

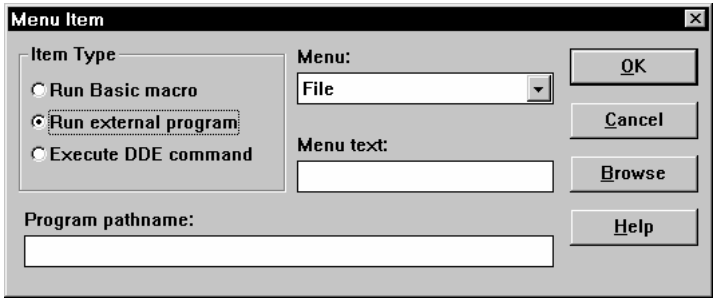
1. **In OMNIC, choose Edit Menu from the Edit menu.**

   The Edit Menu dialog box lists every item in each OMNIC menu and the current state of each item.



2. **Choose Add Item to add the macro to the menus.**

   The Menu Item dialog box appears.



3. **Select Run External Program in the Item Type box.**

4. **Enter the pathname of the Pro macro executable file.**

   Type the complete pathname of your Pro macro (.EXE) file in the Program Pathname box. You can also use the Browse button to locate and select a macro.

5. **Enter the menu information.**

   Select from the Menu drop-down list the menu to which you want the item added, and then type in the Menu Text box the name of the macro exactly as you want it to appear in the menu. If you want to define a letter for choosing the command using keyboard, place the "&" character immediately before the desired letter in the command name. For example, to use the "N" in the macro name "Noise Macro," type "&Noise Macro" in the text box.

6. **Click OK to close the Menu Item dialog box, and then click OK to close the Edit Menu dialog box.**

7. **If you disabled or hid the Save Configuration As command and want to save your changes, click the Save button.**

   (You will not be able to save your changes after closing the Edit Menu dialog box in this case.) The Save Configuration dialog box appears. This is the same dialog box that appears when you choose Save Configuration As from the File menu. For instructions on using the dialog box, see "How to save a configuration" in the "File" chapter of the *OMNIC User's Guide*.

8. **When you are finished adding your Pro macro(s) to OMNIC menus, choose OK.**

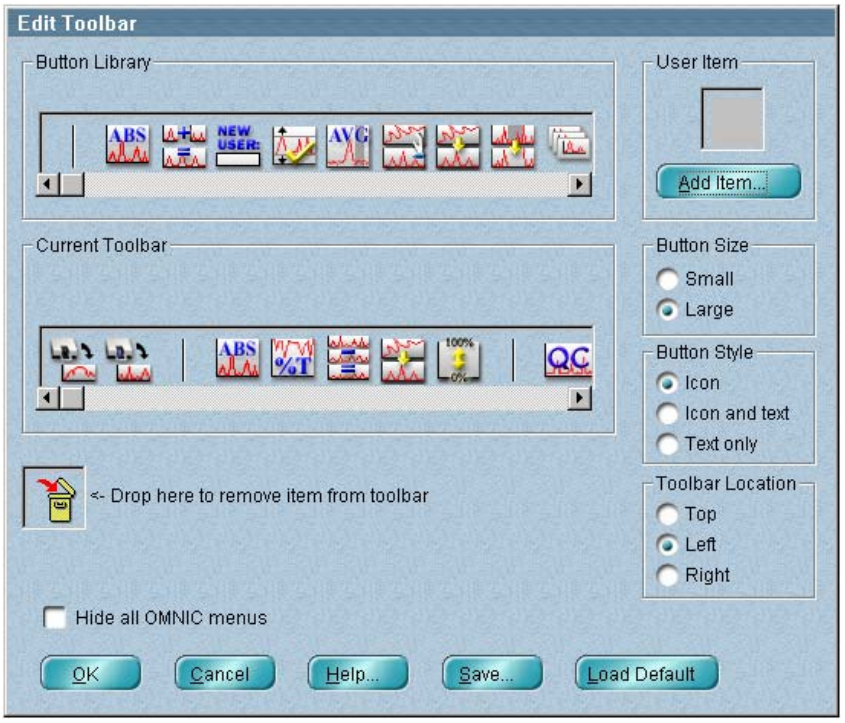
   To save your menu changes, use Save Configuration As in the File menu (if it is available). You will be prompted to save your changes when you exit OMNIC.

Adding a macro to the OMNIC toolbar

The Edit Toolbar command in the OMNIC Edit menu allows you to customize the toolbar by adding buttons to initiate Pro macros. To assign a Pro macro to the OMNIC toolbar, follow these steps:

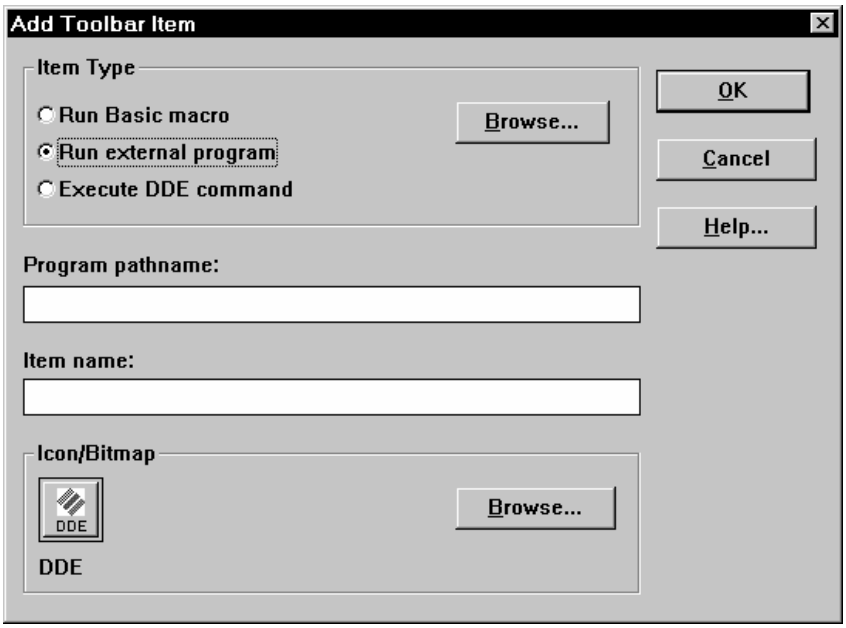
1. **In OMNIC, choose Edit Toolbar from the Edit menu.**

   The Edit Toolbar dialog box appears showing the standard button library and the current toolbar.



2. **Choose Add Item to add the macro to the toolbar.**

   The Add Toolbar Item dialog box appears.



3. **Select Run External Program in the Item Type box.**

4. **Enter the pathname of the Pro macro executable file.**

   Type the complete pathname of your Pro macro (.EXE) file in the Program Pathname box. You can also click the Browse button to display a dialog box that allows you to select a macro.

5. **Enter the button name and icon information.**

   In the Item Name text box type the name you want to assign to the new button. Once the button is added to the toolbar, you can point to the button momentarily to display its assigned name. A default icon for the button appears in the Icon/Bitmap box. To select another icon for the new button, click the Browse button in the Icon/Bitmap box. A dialog box appears allowing you to select a file that contains a bitmap or icon image (if you have them on your system). When you select a file, the image it contains appears as a button. When you are finished selecting a file, click OK. The new button appears in the Icon/Bitmap box.

6. **Click OK to close the Add Toolbar Item dialog box.**

7. **Drag the new button from the User Item box to the desired location in the current toolbar.**

8. **Click OK to close the Edit Toolbar dialog box.**

9. **If you turned on Hide All OMNIC Menus and want to save your toolbar in a configuration file, use the Save button.**

   You will not be able to use Save Configuration As in the File menu to do this, since the menu will not be available. The setting of Hide All OMNIC Menus will also be saved in the configuration. In the Save Configuration dialog box enter a filename for the configuration and then choose Save. Do *not* turn on Set As Default Configuration when you save the configuration.

Note If you have the Val-Q DS™ software, you may be prompted to digitally sign a file during this step. Follow the instructions that appear on the screen. If you are not prompted but want to sign the file, turn on the Use Digital Signature in the Save Configuration dialog box. ▲

10. **When you are finished adding a macro to the toolbar, choose OK.**

    To save your toolbar changes, use Save Configuration As in the File menu.

Calling a Pro macro from Macros\Basic

You can add Pro macros to a Basic macro by using the Macro command in the Insert menu of the optional Macros\Basic software. If you add an executable file to a Macros\Basic macro, the OMNIC Macros application (either Macros\Basic or Macro Panel) will suspend execution of the calling macro until the executable file finishes or tells OMNIC Macros to resume. For more information, see the *OMNIC Macros\Basic User's Guide*.

Follow these steps to add a Pro macro to a Macros\Basic macro:

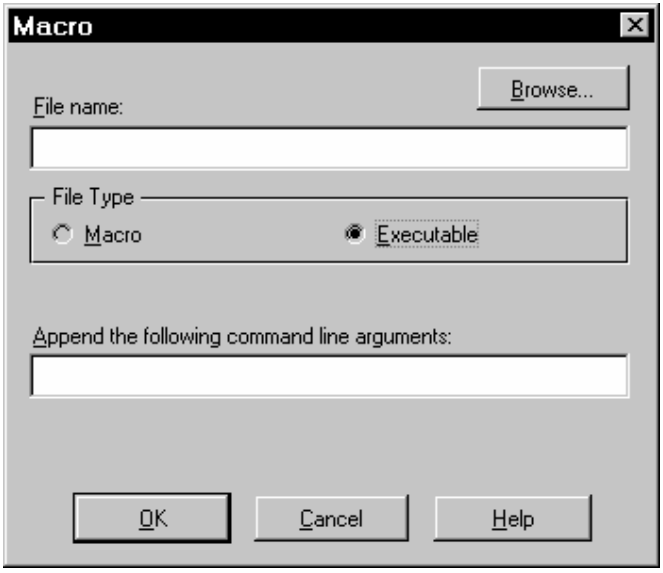
1. **Start the Macros\Basic application.**

   See your Macros\Basic documentation for instructions.

2. **Position the insertion point where you want to add the macro.**

3. **Choose Macro from the Insert menu.**

   The Macro dialog box appears.



4. **Enter the name of the macro.**

   Type the name of the Pro macro in the File Name box, or use the Browse button to display a dialog box that lets you select the macro.

5. **Select the file type.**

   Select Executable in the File Type box to run a Pro macro.

   You can also append a command line argument containing values you want to send to the Pro macro that has been called. For example, you could call the executable file "run.exe" and append the command line argument "abc.txt". This would be the same as using the Windows Run command to execute the file "abc.txt".

6. **Verify the pathname of the macro to use, and then click OK.**

Using Pro macros in Macros\Basic loops

A Macros\Basic macro will start your Visual Basic application when it executes a Pro macro task in the macro task sequence. If your application is already running, the Macros\Basic macro will tell your Visual Basic application to run the Form_Load procedure in your startup form. The Form_Load procedure should carry out any initialization tasks you want to occur each time your application is run.

You can use the GetArgStr OmTalk routine to determine whether or not the application is being run again from the Macros\Basic macro. The GetArgStr routine returns the string "rerun" when your application is being run again by Macros\Basic. (For more information, see the "Using the OmTalk Visual Basic Routines" chapter in the this manual.)

Using the OmTalk Routines

OmTalk is a set of Visual Basic routines designed to simplify the development of macros using Visual Basic and the OMNIC commands and parameters. The OmTalk routines handle the communications between Visual Basic and OMNIC so that you don't have to program the interactions between the two applications.

The OmTalk routines are included on the Macros\Pro software disk in two files: OMTALK.FRM and OMTALK.BAS. OMTALK.FRM contains a text box control that handles the communications. OMTALK.BAS contains the code for the OmTalk routines.

Note The OMTALK16.FRM and OMTALK16.BAS files are also included for use with Visual Basic 3.0. If you are running Visual Basic 3.0, you must use these files because Visual Basic 3.0 does not recognize files that are in Visual Basic 4.0 format. The contents of the OMTALK16 and OMTALK files are identical; only the file formats are different. ▲

The OmTalk routines are listed in the Macros\Pro on-line help system and include the following:

EndOMNIC
ErrMsgBox
ErrOMNIC
ExecuteApp
ExecuteOMNIC
FindOMNICData
GetApp
GetArgStr
GetDataArray
GetItem
GetMVVal
GetOMNIC
GetOMNICName
GetOMNICVersion
GetSpecCollectTime
GetSpecData
GetVal
ItemCount
Pop
ResumeMacro
SetApp
SetDataArray
SetMVVal
SetOMNIC
SetSpecData
StartOMNIC
Strip

Adding OmTalk to Visual Basic 6.0 projects

In order to use OmTalk in your Visual Basic 6.0 projects, you need to do several things. First, since the OmTalk routines handle the DDE communications between OMNIC and Visual Basic, you must include the OmTalk files (OMTALK.FRM and OMTALK.BAS) in each Visual Basic project you create. To add the OmTalk files to your Visual Basic 6.0 project:

1. Open Visual Basic and create a new project.

2. Choose Add File from the Project menu.

3. Select the file OMTALK.FRM.

4. Choose Add File from the File menu again.

5. Select the file OMTALK.BAS.

To have these files loaded at run time, include the statement "Load OmTalk" in the Form_Load event procedure of your startup form. For example:

```
Sub Form_Load ()
  Load OmTalk
End Sub
```

If you are using Sub Main as your startup procedure, add the Load OmTalk statement to Sub Main.

Troubleshooting when using OmTalk

If you have problems using OmTalk within your Visual Basic projects, check your projects for the following items.

1. Make sure you have added the OMTALK.BAS and OMTALK.FRM files to your project. These two files contain the routines that allow you to communicate with OMNIC.

2. Make sure the Form_Load procedure in your startup form contains the line “Load OmTalk”. If you are using Sub Main as your startup procedure, this line must be added to Sub Main.

   The statement “Load OmTalk” loads the form OMTALK.FRM and initializes variables that are used for the DDE conversation with OMNIC. If you attempt to communicate with OMNIC without loading OmTalk, your first communication may have unpredictable results.

Types of OmTalk routines

The OmTalk routines handle the following basic interactions between Visual Basic and OMNIC:

- OMNIC program control
- Parameter control
- Command execution
- Error handling
- Basic macro interaction
- Data array operations

OMNIC program control routines

When the OmTalk subroutines are used in a project, they automatically start the OMNIC application if it is not already running. However, you may want to have the OMNIC application start with a particular window style. For example, you may want the application window to be minimized when it is first started. The StartOMNIC subroutine allows you to specify the window style for OMNIC when it starts. The EndOMNIC subroutine can be used to stop the OMNIC application from within your Visual Basic project.

Parameter control routines

Several OmTalk subroutines can be used to set and read OMNIC parameters. The SetOMNIC subroutine is used to set individual OMNIC parameters. The GetOMNIC function can be used to read OMNIC parameters. The GetVal function can be used to access individual portions of parameter values when GetOMNIC returns a complicated value such as Result Current.

Command execution routines

The ExecuteOMNIC subroutine is used to send commands to OMNIC.

Error handling routines

It is good programming practice to check for error conditions when using DDE. OmTalk provides several error handling routines. The ErrMsgBox and ErrOMNIC subroutines can be used to verify that the other OmTalk subroutine operations have successfully completed. The ErrMsgBox subroutine can be used to display a message box when an error involving OmTalk subroutines occurs. The ErrOMNIC subroutine returns an error value that can be checked for more sophisticated error handling.

Basic macro interaction routines

You can write Pro macros that can be called from within Basic macros created with Macros\Basic. For example, you may want to write a Pro macro that can be used within a loop in a Basic macro. However, once the Pro macro is encountered in the Basic macro, the Basic macro waits to resume operation until it receives a ResumeMacro message from the Pro macro. You can use the ResumeMacro subroutine within the Pro macro to resume executing the Basic macro.

You may also want to know when a Pro macro is being run for the first time within a Basic macro loop. For example, you may want the Pro macro to perform some initialization functions when it is run for the first time within the loop. The GetArgStr subroutine can be used to determine if the Pro macro is being run for the first time within a Basic macro loop or if it is being run again from inside the loop.

Data array routines

Visual Basic includes powerful array functions that can be used to perform mathematical functions on your spectral data files. The GetSpecData and GetDataArray subroutines can be used to extract a portion of an OMNIC spectrum into a Visual Basic array. The data can then be manipulated with the Visual Basic array operations. The SetSpecData and SetDataArray subroutines can then be used to transfer the manipulated data back into an OMNIC spectral window.

Note GetDataArray and SetDataArray cannot be used with 32-bit Visual Basic projects; use the GetSpecData and SetSpecData routines instead. ▲

List of OmTalk routines

A list of the OmTalk routines is shown below. Following this list are descriptions of the routines in alphabetical order.

EndOMNIC
ErrMsgBox
ErrOMNIC
ExecuteApp
ExecuteOMNIC
FindOMNICData
GetApp
GetArgStr
GetDataArray
GetItem
GetMVVAl
GetOMNIC
GetOMNICName
GetOMNICVersion
GetSpecCollectTime
GetSpecData
GetVal
ItemCount
Pop
ResumeMacro
SetApp
SetDataArray
SetMVVal
SetOMNIC
SetSpecData
StartOMNIC
Strip

EndOMNIC

This OmTalk routine causes the OMNIC application to quit.

Syntax: EndOMNIC

Remarks: The EndOMNIC statement does not have any arguments.

Example: This example displays a dialog box with Yes and No buttons. If the Yes button is clicked, it uses EndOMNIC to close OMNIC.

```
If MsgBox("Do you want to close OMNIC?", vbQuestion Or vbYesNo) =
↙vbYes Then
    EndOMNIC
End If
```

ErrMsgBox

This OmTalk routine displays a message in a dialog box if an OMNIC DDE error occurs. This message describes the DDE error that occurred while talking with OMNIC.

Syntax: ErrMsgBox

Remarks: The ErrMsgBox statement does not have any arguments.

Use the ErrMsgBox statement after all OmTalk statements and functions. If the function was unsuccessful, ErrMsgBox displays a message that describes the error. If no error occurs, no message box is displayed.

Example: These examples use ErrMsgBox to display any errors that may result from a noise calculation.

```
SetOMNIC “Display RegionStart”, 2250.0
ErrMsgBox
ExecuteOMNIC “CalculateNoise”
ErrMsgBox
```

ErrOMNIC

This OmTalk routine returns the OmTalk error status.

Description: Returns OmTalk error status.

Syntax: ErrOMNIC()

Remarks: The ErrOMNIC function does not have any arguments.

The function ErrOMNIC returns an integer that is the run-time error code after an OmTalk function or statement. If the procedure was successful, ErrOMNIC returns a value of zero; otherwise, it returns the value of the Visual Basic Err function.

Use the ErrOMNIC function after one of the OmTalk statements or functions to test whether it was successful.

Example: This example uses ErrOMNIC to see if the CalculateNoise command was successfully carried out by OMNIC.

```
ExecuteOMNIC “CalculateNoise”
While ErrOMNIC() <> 0
    'Command failed. Set region and try again.
    SetOMNIC “Display RegionStart”, 2600
    SetOMNIC “Display RegionEnd”, 2400
    ExecuteOMNIC “CalculateNoise”
    ErrMsgBox
End While
```

ExecuteApp

This OmTalk routine sends a DDE command to a Windows application other than OMNIC.

Syntax: ExecuteApp <Application|Topic>, <DDE Command>

Remarks: The ExecuteApp statement uses these arguments:

- Application|Topic - A string expression that is the name of the application, the pipe char | (char code 124), and the topic you want to communicate with. Make sure there are no spaces in this string expression. This is the server or source application referred to in the Visual Basic documentation.
- DDE Command - A variant expression that contains the exact text of the command and associated arguments that you want the source application to execute.

This statement opens a DDE conversation with the source application (server), sends the command, and then closes the DDE conversation.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether ExecuteApp was successful. An integer value of zero is returned if the application carried out the command. If ExecuteApp was not successful, the appropriate Visual Basic DDE error code is returned.

Continue to use the ExecuteOMNIC statement rather than this one if you are working with OMNIC. ExecuteOMNIC is optimized to do a better job of managing the communication link with OMNIC.

Note The source application must be running before the ExecuteApp statement is executed. Unlike ExecuteOMNIC, ExecuteApp will not automatically start the application if it is not running. If the application is not running, the ErrOMNIC function will return the Visual Basic DDE error code 282. ▲

Example: This example uses ExecuteApp to copy the contents of the cell in row one, column one of the active Microsoft Excel® spreadsheet to the contents of the cell in row two, column two.

```
ExecuteApp "Excel|[Book1]Sheet1", "[Copy(""R1C1"", ""R2C2"")]"
ErrMsgBox
```

ExecuteOMNIC

This OmTalk routine sends a command to OMNIC.

Syntax: ExecuteOMNIC <string expression>

Remarks: The ExecuteOMNIC statement takes an OMNIC DDE command as its single argument. This argument must contain the exact text of the command and any associated arguments that you want OMNIC to execute.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether ExecuteOMNIC was successful. ErrOMNIC returns an integer value of zero if OMNIC carried out the command. If the command was not successful, ErrOMNIC returns the Visual Basic DDE error code.

Example: This example uses ExecuteOMNIC to collect a sample spectrum. If a DDE error occurs, it is displayed by ErrMsgBox.

```
ExecuteOMNIC "CollectSample"
ErrMsgBox
```

FindOMNICData

This OmTalk routine retrieves the pathname of the Data directory for the current version of OMNIC.

Syntax: FindOMNICData
(<DataDirectory>,<OmnicName>,<EZOmnicName>)

Remarks: The FindOMNICData statement has three arguments.

- DataDirectory - The root directory for storing OMNIC data.
- OMNICName - The name of the OMNIC application: omnic.exe or omnic32.exe.
- EZOmnicName - The name of the EZ OMNIC application: ezomnic.exe or ezomnic32.exe.

The FindOMNICData command is typically used to retrieve the pathname of the OMNIC Data Directory. This is the root path to all OMNIC directories used to store data, such as OMNIC spectra. For OMNIC 4.1x installations, the path is typically C:\OMNIC. For OMNIC 5.x installations, the path is usually C:\MY DOCUMENTS\OMNIC.

Example: This example uses FindOMNICData to retrieve the pathname of the OMNIC data directory.

```
Dim DataDirectory as string
Dim OmnicName as string
Dim EZOmnicName as string
OmnicName=GetOMNICName()
FindOMNICData (DataDirectory,OmnicName,EZOmnicName)
```

GetApp

This OmTalk routine returns the current value of a parameter or object property in a Windows application other than OMNIC.

Syntax: GetApp(<Application|Topic>, <Parameter>)

Remarks: The GetApp function uses these arguments:

- Application|Topic - A string expression that is the name of the application, the pipe char | (char code 124), and the topic you want to communicate with. Make sure there are no spaces in this string expression. This is the server or source application referred to in the Visual Basic documentation.
- Parameter - A string expression that is the name of the parameter or object property whose value you want to get.

GetApp returns a Variant data type that contains the current value of the requested parameter or object property if the DDE command was successful. If GetApp is unsuccessful, it returns an empty string value (““).

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether GetApp was successful. ErrOMNIC returns an Integer value of zero if the source application supplied the requested parameter or object property value. If the request was not successful, ErrOMNIC returns the appropriate Visual Basic DDE error code.

Continue to use the GetOMNIC function rather than this one if you are working with OMNIC. GetOMNIC is optimized to do a better job of managing the communication link with OMNIC.

Note The application must be running before the GetApp function is executed. Unlike GetOMNIC, GetApp will not automatically start the application if it is not running. If the application is not running, the ErrOMNIC function will return the Visual Basic DDE error code 282. ▲

(Continued on next page)

Example: This example uses GetApp to obtain the current value of the cell in row two, column three of the Microsoft Excel spreadsheet Sheet1 of workbook Book1.

```
Dim lvResult As Variant
lvResult = GetApp("Excel|[Book1]Sheet1", "R2C3")
If ErrOMNIC() = 0 Then
    MsgBox "The value of the cell = " & lvResult
Else
    ErrMsgBox
    End If
```

GetArgStr

This OmTalk routine returns a String containing the argument portion of the command string sent by a destination application to your Visual Basic application in a DDE conversation. (See LinkExecute Event in the Visual Basic documentation.)

Syntax: GetArgStr()

Remarks: The GetArgStr function does not have any arguments. The function GetArgStr returns the String “rerun” when your application is being run again by Macros\Basic or Macro Panel. You can use this function in the Form_Load procedure of your startup form to test whether your application is being rerun. If there is no argument portion in the command string sent by the destination application, GetArgStr returns the command string rather than an empty string.

Example: This example uses GetArgStr to test for reruns. If the application is being rerun, it repeats a calculation. The application is initialized only the first time it is run.

```
Sub Form_Load ()
  If GetArgStr() = “rerun” Then
    'This app is being run again by a Basic Macro; just repeat calculation:
    RepeatCalc
  Else
    'Initial load:
    Load OmTalk
    Init
    RepeatCalc
  End If
End Sub
```

GetDataArray

This OmTalk routine obtains numerical spectral data from the currently selected OMNIC spectrum and places it into an array. This routine can be used only with 16-bit versions of Visual Basic. If you are using 32-bit Visual Basic 4.0 or Visual Basic 5.0, you must use the GetSpecData routine instead.

Syntax: GetDataArray <firstX>, <lastX>

Remarks: The GetDataArray statement uses these arguments:

- firstX - A numerical expression that is one boundary of a spectral region.
- lastX - A numerical expression that is the other boundary of a spectral region.

Both firstX and lastX should have the same unit as the X-axis.

GetDataArray returns the spectral data into the global array DataArray() of type Single. GetDataArray also sets the following global variables of type Variant:

- GetSpecNum – The number of data points returned.
- GetSpecFirstX – The X-axis value of first point in DataArray().
- GetSpecLastX – The X-axis value of last point in DataArray().
- GetSpecIncrement – The data point spacing of DataArray() in X-axis units.

The above variables are declared as global variables of the Variant data type in the OmTalk declarations section. The DataArray() array is declared as a Single data type and global. Do not declare these variables in your code, because these are global variables. Global variables are available in every procedure in every form and code module in your application.

Use the function ErrOMNIC to test whether GetDataArray was successful. ErrOMNIC returns an integer value of zero if OMNIC supplied the requested spectral data. If the request was not successful, ErrOMNIC returns the appropriate Visual Basic DDE error code.

(Continued on next page)

The GetDataArray routine in this version of OmTalk accesses OMNIC spectral data by direct calls into OMNIC rather than via DDE. This results in faster execution than the GetSpecData routine but can be used only with Visual Basic 3.0 or 16-bit Visual Basic 4.0.

The GetDataArray routine is completely compatible with the GetSpecData routine. The only real difference you may notice is that the spectral data is of type Single and put into the global array DataArray() instead of the SpecData() array of type Variant.

Example: See the sample project GETDATA.VBP shipped with the Macros\Pro software for a complete example of GetDataArray. The following example uses GetDataArray to obtain the spectral data from the region 1620 - 1600 cm^{-1}. A message box displays the number of data points returned and their X and Y values.

```
Dim lvText As String
Dim lvCrLf As String
Dim lvTab As String
Dim lvIndex As Integer
Dim lvXVal As Single
GetDataArray 1620, 1600
If ErrOMNIC() = 0 Then
    lvText$ = “Number of data points returned = “ & GetSpecNum
    lvCrLf$ = Chr$(13) & Chr$(10)
    lvTab$ = Chr$(9)
    lvText$ = lvText$ & lvCrLf$ & “Point” & lvTab$ & “X value” & 1vTab$ &
    ↙“Y value”
    For lvIndex% = 1 To GetSpecNum
        lvXVal! = GetSpecFirstX - (lvIndex% - 1) * GetSpecIncrement
        lvText$ = lvText$ & lvCrLf$ & Str$(lvIndex%) & lvTab$ &
        ↙Str$(lvXVal!)
        lvText$ = lvText$ & lvTab$ & Str$(DataArray(lvIndex%))
    Next lvIndex%
    MsgBox lvText$
Else
    ErrMsgBox
End If
```

GetItem

This OmTalk routine returns the value of an item in a list as a Variant data type.

Syntax: GetItem(<list string>, <item number>)

Remarks: The GetItem function uses these arguments:

- list string - A string expression containing a list of separated values. Assumes items are separated by the list separator character specified in the International section of the Windows Control Panel application.
- item number - An integer expression that is the item number in the list you want.

This function (and also the ItemCount and Pop functions) is useful when you want to obtain specific items from the list of values returned by the Result Array OMNIC parameter.

Example: This example obtains the list of results from a noise calculation and picks out the third value, which is the peak-to-peak noise value.

```
Dim lvList As String

Dim lvNoise As Single

ExecuteOMNIC “CalculateNoise”
If ErrOMNIC() = 0 Then
    lvList$ = GetOMNIC(“Result Array”)
    lvNoise! = GetItem(lvList$, 3)
    MsgBox “The peak-to-peak noise level = “ & Format$(lvNoise!, “0.0000”)
Else
    ErrMsgBox
End If
```

GetMVVal

This OmTalk routine returns the current value of a Macros\Basic macro variable.

Syntax: GetMVVal(<Macro variable number>)

Remarks: The GetMVVal function takes a long expression as its single argument. You must specify the number of the macro variable whose value you want to obtain. This value must be between 1 and 65535 for Macros\Basic 4.0 or higher and between 1 and 100 for Macros\Basic 3.0 or lower. Do not include the mv prefix. GetMVVal returns a variant data type that contains the current value of the requested macro variable if the DDE command was successful. If GetMVVal is unsuccessful, it returns an empty string value (““).

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether GetMVVal was successful. ErrOMNIC returns an integer value of zero if the Macros\Basic macro supplied the requested object value. If the request was not successful, ErrOMNIC returns the Visual Basic DDE error code.

Example: This example uses GetMVVal to obtain the current value of macro variable mv3.

```
Dim lvVal As Variant

lvVal = GetMVVal(3)
If ErrOMNIC() = 0 Then
    MsgBox “The current value of mv3 = “ & Format$(lvVal, “0.00”)
Else
    ErrMsgBox
End If
```

GetOMNIC

This OmTalk routine returns the value of an OMNIC parameter.

Syntax: GetOMNIC(<Parameter name>)

Remarks: The GetOMNIC function takes a string expression as its single argument. You must specify the name of the OMNIC parameter whose value you want to obtain. The parameter name argument must contain a group name followed by a space and the parameter name.

GetOMNIC returns a Variant data type that contains the value of the requested parameter if the DDE command was successful. If GetOMNIC is unsuccessful, it returns the text "#ERROR*n*#", where *n* is the Visual Basic error code, Err.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether GetOMNIC was successful. ErrOMNIC returns an integer value of zero if OMNIC supplied the requested parameter value. If the request was not successful, ErrOMNIC returns the Visual Basic DDE error code.

Example: This example uses GetOMNIC to obtain the result of a noise calculation.

```
ExecuteOMNIC "CalculateNoise"
If ErrOMNIC() = 0 Then
    noise$ = GetOMNIC("Result Current")
Else
    ErrMsgBox
End If
```

GetOMNICName

This OmTalk routine retrieves the full application name from OMNIC. Use it to determine which version of OMNIC (16- or 32-bit) or EZ OMNIC (16- or 32-bit) is currently running.

Syntax: GetOMNICName ()

Remarks: The GetOMNICName statement has no argument.

Example: This example uses GetOMNICName to retrieve the name of the current version of OMNIC.

```
dim szOmnicName as string
szOmnicName=GetOMNICName ()
```

GetOMNICVersion

This OmTalk routine returns the version number of OMNIC as a variant.

Syntax: GetOMNICVersion (<avFormat>)

Remarks: The GetOMNICVersion statement has a single argument which can be used to control the format of the returned version number.

- avFormat = 0 - Returns the version number in native format with major and minor revisions separated by decimals. Example: 5.0.0.1
- avFormat = 1 - Returns the version number as a floating point number with minor revisions concatenated. Example: 5.001

Example: This example uses GetOMNICVersion to retrieve the version number in the native format.

```
dim lvVal as Variant
lvVal=GetOMNICVersion (0)
```

GetSpecCollectTime

This OmTalk routine returns a variant that indicates the date and time when the currently selected spectrum was collected.

Syntax: GetSpecCollectTime(<format>)

Remarks: The argument <format> is an integer that determines the format of the returned date and time as follows.

- 1 - Returns a variant of VarType 7 (Date) containing a date and time stored internally as a double-precision number. This is referred to as a time serial number in the Visual Basic documentation. Numbers to the left of the decimal point represent the date; numbers to the right represent the time. See the Visual Basic Now function for details on this format.
- 2 - Returns a variant of VarType 8 (String) containing the date and time in the same format as shown in the OMNIC spectrum collection and processing information.

The time serial form (<format> = 1) is the time/date representation used in Visual Basic. You can use this number with the Visual Basic Format function to render time and date in any format you want. For more details on using the time serial form, see the documentation on the following Visual Basic functions: Day, Hour, Minute, Month, Now, Second, Weekday and Year.

Example: The first example uses GetSpecCollectTime to get the spectrum collection time in time serial format. The Format function is used to display the result as “14 January 93 2:23 pm”. The second example uses GetSpecCollect-Time to get the spectrum collection time as a string and display it as “Thu Jan 14 14:24:20 1993”.

```
Dim lvTime As Variant

lvTime = GetSpecCollectTime(1)
MsgBox Format(lvTime, “d mmmm yy h:mm am/pm”)

lvTime = GetSpecCollectTime(2)
MsgBox lvTime
```

GetSpecData

This OmTalk routine obtains numeric spectral data from an OMNIC spectrum and places it in an array. For 32-bit Visual Basic projects, you must use this routine instead of the GetDataArray routine. If you are building a 16-bit Visual Basic application, you may use the GetDataArray routine instead; it is faster.

Syntax: GetSpecData <firstX>, <lastX>

Remarks: The GetSpecData statement uses these arguments:

- firstX - A numeric expression that is the one boundary of a spectral region.
- lastX - A numeric expression that is the other boundary of a spectral region.

Both firstX and lastX should have the same unit as the X-axis.

GetSpecData returns the spectral data into the global array SpecData(). GetSpecData also sets the following global variables:

- GetSpecNum - The number of data points returned.
- GetSpecFirstX - The X-axis value of the first point in SpecData().
- GetSpecLastX - The X-axis value of the last point in SpecData().
- GetSpecIncrement - The data point spacing of SpecData() in X-axis units.

The above variables and the SpecData() array are defined in the OmTalk declarations section as variant data types. You do not need to define these variables in your code.

Use the function ErrOMNIC to test whether GetSpecData was successful. ErrOMNIC returns an integer value of zero if OMNIC supplied the requested spectral data. If the request was not successful, ErrOMNIC returns the Visual Basic DDE error code.

(Continued on next page)

Example: See the sample project GetData.vbp shipped with the Macros\Pro software for a complete example of GetDataArray.

The following example uses GetSpecData to obtain the spectral data from the region from 1620 to 1600 cm^{-1}. A message box displays the number of data points returned and the value of the first data point.

```
GetSpecData 1620, 1600
if ErrOMNIC() = 0 Then
    tex$ = “Number of data points returned = “ + GetSpecNum
    tex$ = tex$ + Chr$(13) & Chr$(10) & “1st point X value =”
    ↙& GetSpecFirstX
    tex$ = tex$ & Chr$(13) & Chr$(10) & “1st point Y value =”
    ↙& SpecData(1)
    MsgBox tex$
Else
    ErrMsgBox
End If
```

GetVal

This OmTalk routine returns the numeric value corresponding to the word following the search string in a text string. This function has been made obsolete by use of the GetItem function in conjunction with the OMNIC parameter Result Array. See the Get Item function.

Syntax: GetVal(<text string> <search string>)

Remarks: The GetVal function uses these arguments:

- text string - The string expression being searched.
- search string - The string expression being sought.

The GetVal function always returns a Double data type. If the search string is not found in the text string, a value of zero is returned. GetVal ignores any ":" or "(" characters that may occur between the search string and the value that follows it.

The format for OMNIC text returned via the Result Current parameter is the same as that shown in the OMNIC readout or dialog boxes. For example, the CorrectedPeakArea command sets Result Current to "Area: 19.289 Uncorrected: 25.157 Region: (1468.346, 1423.154) Baseline: (1468.846, 1415.667)".

Example: This example uses GetVal to extract the values of the minimum and maximum from the result string of the OMNIC MinMax command. Note that the colon following the search string is optional.

```
ExecuteOMNIC "minmax"
ErrMsgBox
result$ = GetOMNIC("Result Current")
ErrMsgBox
min = GetVal(result$, "Min:")
max = GetVal(result$, "Max")
```

ItemCount

This OmTalk routine returns the number of items in a list as an integer data type.

Syntax: ItemCount(<list string>)

Remarks: The ItemCount function takes a string expression as its single argument. This expression is a list of separated values. ItemCount assumes items are separated by the list separator character specified on the Number tab of the Regional Settings application, located in the Windows Control Panel folder. For Windows 3.1, use the International section of the Control Panel application. ItemCount returns an integer data type that is the number of items in the list string. If the list string argument is empty, ItemCount returns a value of zero. This function (and also the GetItem and Pop functions) is useful when you want to obtain specific items from the list of values returned by the Result Array OMNIC parameter.

Example: This example displays a table of results from a Find Peaks operation.

```
  Dim lvList As String
  Dim lvTable As String
  Dim lvCount As Integer
  Dim lvIndex As Integer
  Dim lvPeakPosition As Single
  Dim lvPeakValue As Single

ExecuteOMNIC "PeakPick 0.25 50"
  If ErrOMNIC() = 0 Then
    lvList$ = GetOMNIC("Result Array")
    lvCount% = ItemCount(lvList$)
    For lvIndex% = 5 To lvCount% Step 2
      lvPeakPosition! = GetItem(lvList$, lvIndex%)
      lvPeakValue! = GetItem(lvList$, lvIndex% + 1)
      lvTable$ = lvTable$ & Format$(lvPeakPosition!, "0.00") & Chr$(9)
      lvTable$ = lvTable$ & Format$(lvPeakValue!, "0.0000") & Chr$(13)
      ↙& Chr$(10)
    Next lvIndex%
    MsgBox lvTable$, 64, "Find Peaks Result"
Else
  ErrMsgBox
End If
```

Pop

This OmTalk routine returns the first item in a list and removes this item from the list.

Syntax: Pop(<list string>)

Remarks: The Pop function takes a string expression as its single argument. This expression is a list of separated values. Pop assumes items are separated by the list separator character specified on the Number tab of the Regional Settings application, located in the Windows Control Panel folder. For Windows 3.1, use the International section of the Control Panel application.

Pop returns a variant data type that is the value of the first item in the list string. If the list string argument is empty, Pop returns a null string value (““).

This function is useful when you want to process items from the list of values returned by the Result Array OMNIC parameter. See also the functions ItemCount and GetItem.

Example: This example stores the results of a Quantify operation in the array lvConc. Note how the array lvConc is dynamically allocated. This lets this code work for any number of results up to 40 components.

```
Dim lvQuantResult As String
    Dim lvConc() As Single
    Dim lvVal As Variant
    Dim lvIndex As Integer

ExecuteOMNIC “Quantify”
    lvQuantResult$ = GetOMNIC(“Result Array”)
    If ErrOMNIC() = 0 Then
        ReDim lvConc!(40)
        lvIndex% = 0
        lvVal = Pop(lvQuantResult$)
    While lvVal <> ““
        lvIndex% = lvIndex% + 1
        lvConc!(lvIndex%) = lvVal
        lvVal = Pop(lvQuantResult$)
    Wend
    ReDim Preserve lvConc!(lvIndex%)
End If
```

ResumeMacro

This OmTalk routine tells either the Macros\Basic or Macro Panel application to resume executing tasks in a Macros\Basic macro.

Syntax: ResumeMacro

Remarks: The ResumeMacro statement does not have any arguments. Use it to control when to continue with execution of tasks in a Macros\Basic macro.

When you run a macro from the OMNIC toolbar or the Macro Panel or use the Test command from Macros\Basic, the Macros\Basic macro suspends execution at the Macro task that executes your Visual Basic application. Use ResumeMacro to continue execution of the Macros\Basic tasks following the Macro task in your macro. If you do not call ResumeMacro, your macro will remain suspended until your Visual Basic application closes.

If a macro does not need to wait for your Visual Basic application, use the ResumeMacro statement in the Load_Form event procedure of your Visual Basic startup form.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether ResumeMacro was successful. ErrOMNIC returns an integer value of zero if either of the applications “Macros\Basic” or “Macro Panel” carried out the request. If the request was not successful, ErrOMNIC returns the Visual Basic error code.

Example: This example uses ResumeMacro to resume a macro after it has displayed the result of a noise calculation.

```
ExecuteOMNIC “CalculateNoise”
ErrMsgBox
noise$ = GetOMNIC(“Result Current”)
ErrMsgBox
MsgBox noise$, 64, “Noise Result”
ResumeMacro
ErrMsgBox
```

SetApp

This OmTalk routine sets the value of a parameter or object property in a Windows application other than OMNIC.

Syntax: SetApp <Application|Topic>, <Parameter>, <Value>)

Remarks: The SetApp statement uses these arguments:

- Application|Topic - A string expression that is the name of the application, the pipe char | (char code 124), and the topic you want to communicate with. Make sure there are no spaces in this string expression. This is the server or source application referred to in the Visual Basic documentation.
- Parameter - A string expression that is the name of the parameter or object property whose value you want to set.
- Value - A variant expression that is the value you want to assign to the parameter or object property.

This statement opens a DDE conversation with the source application, sets the parameter or object property value, and then closes the DDE conversation.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether SetApp was successful. ErrOMNIC returns an integer value of zero if the application set the parameter or object property to the specified value. If SetApp was not successful, ErrOMNIC returns the appropriate Visual Basic DDE error code.

Continue to use the SetOMNIC statement rather than this one if you are working with OMNIC. SetOMNIC is optimized to do a better job of managing the communication link with OMNIC.

Note The application must be running before the SetApp statement is executed. Unlike SetOMNIC, SetApp will not automatically start the application if it is not running. If the application is not running, the ErrOMNIC function will return the Visual Basic DDE error code 282. ▲

(Continued on next page)

Example: This example sets the value of the cell in row two, column three to 0.153 in the Microsoft Excel spreadsheet with the name Sheet1. If Excel is not running, the Visual Basic Shell function is used to start Excel.

```
Dim lvHwnd As Integer

SetApp "Excel|[Book1]Sheet1", "R2C3", .153
If ErrOMNIC() <> 0 Then
    If ErrOMNIC() = 282 Then
        lvHwnd% = Shell("excel.exe", 1)
        DoEvents
        SetApp "Excel|[Book1]Sheet1", "R2C3", .153
    Else
        ErrMsgBox
    End If
End If
```

SetDataArray

This OmTalk routine sets the numeric values of spectral data in the currently selected OMNIC spectrum. This routine can be used only with 16-bit versions of Visual Basic. If you are using 32-bit Visual Basic 4.0 or Visual Basic 5.0, you must use SetSpecData instead.

Syntax: SetDataArray

Remarks: The SetDataArray statement does not have any arguments. It sets the spectral data in the currently selected OMNIC spectrum to the values contained into the global array DataArray(). The following global variables specify the number of data points and region of data to be set:

- GetSpecNum - The number of data points returned.
- GetSpecFirstX - The X-axis value of the first point in DataArray().
- GetSpecLastX - The X-axis value of the last point in DataArray().
- GetSpecIncrement - The data point spacing of DataArray() in X-axis units.

The above variables are declared as global variables of Variant data type in the OmTalk declarations section. The DataArray() array is declared as a Single data type and global. Do not declare these variables in your code, because these are global variables. Global variables are available in every procedure in every form and code module in your application.

Use the function ErrOMNIC to test whether SetDataArray was successful. ErrOMNIC returns an integer value of zero if OMNIC accepted the spectral data. If SetDataArray was not successful, ErrOMNIC returns the appropriate Visual Basic DDE error code.

The SetDataArray routine in this version of OmTalk accesses OMNIC spectral data by direct calls into OMNIC rather than via DDE. This results in faster execution than the SetSpecData routine but can be used only with Visual Basic 3.0 or 16-bit Visual Basic 4.0.

(Continued on next page)

The SetDataArray routine is completely compatible with the SetSpecData routine. The main differences you may encounter are:

- The spectral data is stored in the global array DataArray(), which is a Single data type. SetSpecData used the type Variant array SpecData().
- The SetSpecData routine interpolates additional data points if GetSpecNum, the number of data points, is fewer than the number of data points OMNIC needs for the region specified by GetSpecFirstX and GetSpecLastX.

  The SetDataArray will not do this. This is not a problem if you always use GetDataArray in conjunction with SetDataArray. GetDataArray automatically sets GetSpecNum to the correct value for the region you are working with.

Example: See the sample project GetData.vbp shipped with the Macros\Pro software for a complete example of SetDataArray.

The example below uses SetDataArray to fit a straight line across the spectral data from the region 2400 to 2200 cm^{-1}.

```
Dim lvIncrement as Single
Dim lvIndex as Integer

GetDataArray 2400, 2200
lvIncrement! = (DataArray(GetSpecNum) - DataArray(1))/(GetSpecNum - 1)
For lvIndex% = 2 To GetSpecNum
    DataArray(lvIndex%) = DataArray(1) + lvIncrement! * (lvIndex% - 1)
Next lvIndex%
SetDataArray
```

SetMVVal

This OmTalk routine sets the value of a Macros\Basic macro variable.

Syntax: SetMVVal(<Macro variable number>, <Value>)

Remarks: The SetMVVal statement uses these arguments:

- Macro variable number - A long expression that is the number of the macro variable whose value you want to set. Do not include the mv prefix.
- Value - A variant expression that is the value you want to assign to the macro variable.

The macro variable number must be between 1 and 65535 for Macros\Basic 4.0 or higher or between 1 and 100 for Macros\Basic 3.0 or lower.

It is not necessary for the macro variable to be declared in the Macros\Basic macro, since this routine will create the macro variable definition. This is useful when you want to pass values from your Visual Basic application back to a Macros\Basic macro.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether SetMVVal was successful. ErrOMNIC returns an integer value of zero if the Macros\Basic macro set the macro variable to the specified value. If SetMacro was not successful, ErrOMNIC returns the Visual Basic DDE error code.

(Continued on next page)

Example: This example evaluates the first component value from a Quant result and then uses SetMVVal to set the value of macro variable mv3 to an appropriate text message.

```
Dim lvQuantResult As String
Dim lvText As String

ExecuteOMNIC "Quantify"
lvQuantResult$ = GetOMNIC("Result Array")
If ErrOMNIC() = 0 Then
    If .5 < Pop(lvQuantResult$) < 1# Then
        lvText$ = "Product is within acceptable limits."
    Else
        lvText$ = "Product is out of spec."
    End If
    SetMVVal 3, lvText$
End If
ErrMsgBox
```

SetOMNIC

This OmTalk routine sets an OMNIC parameter to a value that you specify.

Syntax: SetOMNIC <Parameter name>, <Value>

Remarks: The SetOMNIC statement uses these arguments:

- Parameter name - A string expression that is the name of an OMNIC parameter whose value you want to set. The parameter name argument must contain a group name followed by a space and the parameter name.
- Value - A variant expression that is the value to set the parameter to.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether SetOMNIC was successful. ErrOMNIC returns an integer value of zero if OMNIC set the parameter to the specified value. If SetOMNIC was not successful, ErrOMNIC returns the Visual Basic DDE error code.

Example: This example uses SetOMNIC to set the spectral region before a noise calculation. It then obtains the result of the noise calculation.

```
SetOMNIC “Display RegionStart”, 2250.0
ErrMsgBox
SetOMNIC “Display RegionEnd”, 2200.0
ErrMsgBox
ExecuteOMNIC “CalculateNoise”
If ErrOMNIC() = 0 Then
    noise$ = GetOMNIC(“Result Current”)
Else
    ErrMsgBox
End If
```

SetSpecData

This OmTalk routine sets the numeric values of spectral data in an OMNIC spectrum. For 32-bit Visual Basic projects, you must use this routine instead of the SetDataArray routine. If you are building a 16-bit Visual Basic application, you may use the SetDataArray routine instead; it is faster.

Syntax: SetSpecData

Remarks: The SetSpecData statement does not have any arguments.

SetSpecData sets the spectral data in the currently selected OMNIC spectrum to the values contained in the global array SpecData(). The following global variables specify the number of data points and region of data to be set:

- GetSpecNum - The number of data points to be set.
- GetSpecFirstX - The X-axis value of the first point in SpecData().
- GetSpecLastX - The X-axis value of the last point in SpecData().

The above variables and the SpecData() array are defined in the OmTalk Declarations section as Variant data types. You do not need to define these variables in your code.

Use the function ErrOMNIC to test whether SetSpecData was successful. ErrOMNIC returns an integer value of zero if OMNIC supplied the requested spectral data. If the request was not successful, ErrOMNIC returns the Visual Basic DDE error code.

Example: See the sample project GetData.vbp shipped with the Macros\Pro software for a complete example of SetSpecData.

This example uses SetSpecData to fit a straight line across the spectral data from the region 2400 to 2200 cm^{-1}.

```
GetSpecData 2400, 2200
increment = (SpecData(GetSpecNum) - SpecData(1)) / (GetSpecNum-1)
For i% = 2 To GetSpecNum
    SpecData(i%) = SpecData(1) + increment*(i%-1)
Next i%
SetSpecData
```

StartOMNIC

This OmTalk routine runs OMNIC if it is not already running.

Note You do not need to call StartOMNIC before using the other OmTalk routines. If OMNIC is not already running when you try to communicate with it, it will be started automatically. Use StartOMNIC *only* if you must launch OMNIC with a specific window style or argument string. If OMNIC is already running when you call StartOMNIC, you may get an error message. ▲

Syntax: StartOMNIC(<Window style> <Argument string>)

Remarks: The StartOMNIC function has these arguments:

- window style - An integer corresponding to the style of the OMNIC window.
- argument string - A string expression containing any arguments or command line switches.

The following table identifies the possible values for the window style and the style of window that occurs as a result:

- 1 - The normal window with focus.
- 2 - Minimized with focus.
- 3 - Maximized with focus.
- 4 - The normal window without focus.
- 7 - Minimized without focus.

Set the argument string argument to an empty string if you do not use any command line options. Command line options are:

- <filename> - The name of a valid spectral data file that is to be displayed in the initial window.
- -b<benchfile> - Use benchfile instead of real bench.
- -i - Run in invisible mode; that is, with no user interface.
- -l <x,y> - The location of the upper-left corner, in pixels.
- -s<width,height> - The width and height, in pixels.

(Continued on next page)

If the StartOMNIC function is successful in starting OMNIC, it returns a variant task identification (ID) for OMNIC. The task ID is a unique number that identifies OMNIC. If StartOMNIC is unsuccessful in starting OMNIC, Visual Basic generates an error message.

Example: These examples use StartOMNIC to launch OMNIC. In the first example, the OMNIC window is minimized to an icon and the Visual Basic application retains the focus. In the second example, the OMNIC window is displayed in the upper-left corner of the screen, 820 wide by 520 high.

```
i = StartOMNIC(7, "")
i = StartOMNIC(1, "-10,0 -s820,520")
```

Note Earlier versions of the manual said the return type was integer. This is incorrect; it should be variant. For 16-bit versions of OMTALK, using the integer return type worked. For 32-bit versions, you must use the variant. You will get overflow error messages if you use an integer type. ▲

Strip

This OmTalk routine returns a string with any leading and trailing white space characters removed.

Syntax: Strip (<text string>)

Remarks: The Strip function takes a text string as its single argument. The function removes any carriage return, line feed, tab or space characters from the start and end of the string, and then returns the text string.

Example: This example removes white space characters from the start and end of a text string but leaves embedded white space alone.

```
Dim lvTex as String
Dim lvRetVal as String
lvTex$ = "  This is "&Chr$(9) & "Column 1" & Chr$(13) & Chr$(10)
lvRetVal = Strip(lvTex$)
```

Using the OmTalk.NET Routines

OmTalk.NET is a set of Visual Basic routines designed to simplify the development of macros using Visual Basic.NET and the OMNIC commands and parameters. The OmTalk routines handle the communications between Visual Basic and OMNIC so that you don't have to program the interactions between the two applications.

The OmTalk.NET routines are included on the Macros\Pro software disk in two files: OMTALK.EXE and OMTALK32.VB. (If you are using Visual Basic 6.0, you must use OMTALK32.BAS.)

The OmTalk routines are listed in the Macros\Pro on-line help system and include the following:

EndOMNIC
ErrMsgBox
ErrOMNIC
ExecuteApp
ExecuteOMNIC
FindOMNICData
GetApp
GetItem
GetMVVal
GetOMNIC
GetOMNICName
GetOMNICVersion
GetSpecCollectTime
GetSpecData
GetVal
ItemCount
LoadOmTalk
Pop
SetApp
SetMVVal
SetOMNIC
SetSpecData
StartOMNIC
Strip
UnloadOmTalk

Note The GetArgStr, GetDataArray, ResumeMacro, and SetDataArray OmTalk routines are not supported by OmTalk.NET. ▲

Adding OmTalk to Visual Basic.NET projects

In order to use OmTalk.NET in your Visual Basic.NET projects, you need to do several things. First, since the OmTalk routines handle the communications between OMNIC and Visual Basic, you must include the OMTALK32.VB in each project you create.

1. Open Visual Basic.NET and create a new project.
2. Choose Add Existing File from the Project menu.
3. Select the file OMTALK32.VB.

To have the OmTalk.NET files loaded at run time, include the statement "LoadOmTalk" in the Form_Load event procedure of your startup form. For example:

```
Sub Form_Load ()
  LoadOmTalk
End Sub
```

If you are using Sub Main as your startup procedure, add the LoadOmTalk statement to Sub Main.

Types of OmTalk.NET routines

The OmTalk.NET routines handle the following basic interactions between Visual Basic.NET and OMNIC:

- OMNIC program control
- Parameter control
- Command execution
- Error handling
- Basic macro interaction
- Data array operations

OMNIC program control routines

When the OmTalk subroutines are used in a project, they automatically start the OMNIC application if it is not already running. However, you may want to have the OMNIC application start with a particular window style. For example, you may want the application window to be minimized when it is first started. The StartOMNIC subroutine allows you to specify the window style for OMNIC when it starts. The EndOMNIC subroutine can be used to stop the OMNIC application from within your Visual Basic project.

Parameter control routines

Several OmTalk subroutines can be used to set and read OMNIC parameters. The SetOMNIC subroutine is used to set individual OMNIC parameters. The GetOMNIC function can be used to read OMNIC parameters. The GetVal function can be used to access individual portions of parameter values when GetOMNIC returns a complicated value such as Result Current.

Command execution routines

The ExecuteOMNIC subroutine is used to send commands to OMNIC.

Error handling routines

It is good programming practice to check for error conditions. OmTalk provides several error handling routines. The ErrMsgBox and ErrOMNIC subroutines can be used to verify that the other OmTalk.NET subroutine operations have successfully completed. The ErrMsgBox subroutine can be used to display a message box when an error involving OmTalk subroutines occurs. The ErrOMNIC subroutine returns an error value that can be checked for more sophisticated error handling.

Data array routines

Visual Basic includes powerful array functions that can be used to perform mathematical functions on your spectral data files. The GetSpecData subroutine can be used to extract a portion of an OMNIC spectrum into a Visual Basic array. The data can then be manipulated with the Visual Basic array operations. The SetSpecData subroutine can then be used to transfer the manipulated data back into an OMNIC spectral window.

List of OmTalk.NET routines

A list of the OmTalk.NET routines is shown below. Following this list are descriptions of the routines in alphabetical order.

EndOMNIC
ErrMsgBox
ErrOMNIC
ExecuteApp
ExecuteOMNIC
FindOMNICData
GetApp
GetItem
GetMVVAl
GetOMNIC
GetOMNICName
GetOMNICVersion
GetSpecCollectTime
GetSpecData
GetVal
ItemCount
LoadOmTalk
Pop
SetApp
SetMVVal
SetOMNIC
SetSpecData
StartOMNIC
Strip
UnloadOmTalk

EndOMNIC

This OmTalk routine causes the OMNIC application to quit.

Syntax: EndOMNIC()

Remarks: The EndOMNIC statement does not have any arguments.

Example: This example displays a dialog box with Yes and No buttons. If the Yes button is clicked, it uses EndOMNIC to close OMNIC.

```
If MsgBox("Do you want to close OMNIC?", vbQuestion Or vbYesNo) =
↙vbYes Then
    EndOMNIC()
End If
```

ErrMsgBox

This OmTalk routine displays a message in a dialog box if an error occurs. This message describes the error that occurred while talking with OMNIC.

Syntax: ErrMsgBox()

Remarks: The ErrMsgBox statement does not have any arguments.

Use the ErrMsgBox statement after all OmTalk statements and functions. If the function was unsuccessful, ErrMsgBox displays a message that describes the error. If no error occurs, no message box is displayed.

Example: These examples use ErrMsgBox to display any errors that may result from a noise calculation.

```
SetOMNIC (“Display RegionStart”, 2250.0)
ErrMsgBox()
ExecuteOMNIC (“CalculateNoise”)
ErrMsgBox()
```

ErrOMNIC

This OmTalk routine returns the OmTalk error status.

Description: Returns OmTalk error status.

Syntax: ErrOMNIC()

Remarks: The ErrOMNIC function does not have any arguments.

The function ErrOMNIC returns an integer that is the run-time error code after an OmTalk function or statement. If the procedure was successful, ErrOMNIC returns a value of zero; otherwise, it returns the value of the Visual Basic Err function.

Use the ErrOMNIC function after one of the OmTalk statements or functions to test whether it was successful.

Example: This example uses ErrOMNIC to see if the CalculateNoise command was successfully carried out by OMNIC.

```
ExecuteOMNIC “CalculateNoise”
While ErrOMNIC() <> 0
    'Command failed. Set region and try again.
    SetOMNIC (“Display RegionStart”, 2600)
    SetOMNIC (“Display RegionEnd”, 2400)
    ExecuteOMNIC (“CalculateNoise”)
    ErrMsgBox()
End While
```

ExecuteApp

This OmTalk routine sends a DDE command to a Windows application other than OMNIC.

Syntax: ExecuteApp (<Application|Topic>, <DDE Command>)

Remarks: The ExecuteApp statement uses these arguments:

- Application|Topic - A string expression that is the name of the application, the pipe char | (char code 124), and the topic you want to communicate with. Make sure there are no spaces in this string expression. This is the server or source application referred to in the Visual Basic documentation.
- DDE Command - A string expression that contains the exact text of the command and associated arguments that you want the source application to execute.

This statement opens a DDE conversation with the source application (server), sends the command, and then closes the DDE conversation.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether ExecuteApp was successful. An integer value of zero is returned if the application carried out the command. If ExecuteApp was not successful, the appropriate Visual Basic error code is returned.

Continue to use the ExecuteOMNIC statement rather than this one if you are working with OMNIC. ExecuteOMNIC is optimized to do a better job of managing the communication link with OMNIC.

Note The source application must be running before the ExecuteApp statement is executed. Unlike ExecuteOMNIC, ExecuteApp will not automatically start the application if it is not running. If the application is not running, the ErrOMNIC function will return the Visual Basic error code 282. ▲

Example: This example uses ExecuteApp to copy the contents of the cell in row one, column one of the active Microsoft Excel® spreadsheet to the contents of the cell in row two, column two.

```
ExecuteApp ("Excel|[Book1]Sheet1", "[Copy(""R1C1"", ""R2C2"")]")
ErrMsgBox()
```

ExecuteOMNIC

This OmTalk routine sends a command to OMNIC.

Syntax: ExecuteOMNIC (<string expression>)

Remarks: The ExecuteOMNIC statement takes an OMNIC command as its single argument. This argument must contain the exact text of the command and any associated arguments that you want OMNIC to execute.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether ExecuteOMNIC was successful. ErrOMNIC returns an integer value of zero if OMNIC carried out the command. If the command was not successful, ErrOMNIC returns the Visual Basic error code.

Example: This Visual Basic.NET example uses ExecuteOMNIC to collect a sample spectrum. If an error occurs, it is displayed by ErrMsgBox.

```
ExecuteOMNIC ("CollectSample")
ErrMsgBox()
```

Example: This Visual Basic 6.0 example uses ExecuteOMNIC to collect a sample spectrum. If an error occurs, it is displayed by ErrMsgBox. Note that the parameters are not enclosed in parentheses when using Visual Basic 6.0.

```
ExecuteOMNIC "CollectSample"
ErrMsgBox
```

FindOMNICData

This OmTalk routine retrieves the pathname of the Data directory for the current version of OMNIC.

Syntax: FindOMNICData
(<DataDirectory>,<OmnicName>,<EZOmnicName>)

Remarks: The FindOMNICData statement has three arguments.

- DataDirectory - The root directory for storing OMNIC data.
- OMNICName - The name of the OMNIC application, omnic32.exe for example.
- EZOmnicName - The name of the EZ OMNIC application ezomnic32.exe for example.

The FindOMNICData command is typically used to retrieve the pathname of the OMNIC Data Directory. This is the root path to all OMNIC directories used to store data, such as OMNIC spectra. For OMNIC 6.x installations, the path is usually C:\MY DOCUMENTS\OMNIC.

Example: This example uses FindOMNICData to retrieve the pathname of the OMNIC data directory.

```
Dim DataDirectory as string
Dim OmnicName as string
Dim EZOmnicName as string
FindOMNICData (DataDirectory,OmnicName,EZOmnicName)
```

GetApp

This OmTalk routine returns the current value of a parameter or object property in a Windows application other than OMNIC.

Syntax: GetApp(<Application|Topic>, <Parameter>)

Remarks: The GetApp function uses these arguments:

- Application|Topic - A string expression that is the name of the application, the pipe char | (char code 124), and the topic you want to communicate with. Make sure there are no spaces in this string expression. This is the server or source application referred to in the Visual Basic documentation.
- Parameter - A string expression that is the name of the parameter or object property whose value you want to get.

GetApp returns a string data type that contains the current value of the requested parameter or object property if the command was successful. If GetApp is unsuccessful, it returns an empty string value (““).

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether GetApp was successful. ErrOMNIC returns an Integer value of zero if the source application supplied the requested parameter or object property value. If the request was not successful, ErrOMNIC returns the appropriate Visual Basic error code.

Continue to use the GetOMNIC function rather than this one if you are working with OMNIC. GetOMNIC is optimized to do a better job of managing the communication link with OMNIC.

Note The application must be running before the GetApp function is executed. Unlike GetOMNIC, GetApp will not automatically start the application if it is not running. If the application is not running, the ErrOMNIC function will return the Visual Basic error code 282. ▲

(Continued on next page)

Example: This example uses GetApp to obtain the current value of the cell in row two, column three of the Microsoft Excel spreadsheet Sheet1 of workbook Book1.

```
Dim lvResult As String
lvResult = GetApp("Excel|[Book1]Sheet1", "R2C3")
If ErrOMNIC() = 0 Then
    MsgBox ("The value of the cell = " & lvResult)
Else
    ErrMsgBox
End If
```

GetItem

This OmTalk routine returns the value of an item in a list as a String data type.

Syntax: GetItem(<list string>, <item number>)

Remarks: The GetItem function uses these arguments:

- list string - A string expression containing a list of separated values. Assumes items are separated by the list separator character specified in the International section of the Windows Control Panel application.
- item number - An integer expression that is the item number in the list you want.

This function (and also the ItemCount and Pop functions) is useful when you want to obtain specific items from the list of values returned by the Result Array OMNIC parameter.

Example: This example obtains the list of results from a noise calculation and picks out the third value, which is the peak-to-peak noise value.

```
Dim lvList As String

Dim lvNoise As String

ExecuteOMNIC (“CalculateNoise”)
If ErrOMNIC() = 0 Then
    lvList$ = GetOMNIC(“Result Array”)
    lvNoise = GetItem(lvList$, 3)
    MsgBox “The peak-to-peak noise level = “ & Format$(lvNoise, “0.0000”)
Else
    ErrMsgBox
End If
```

GetMVVal

This OmTalk routine returns the current value of a Macros\Basic macro variable.

Syntax: GetMVVal(<Macro variable number>)

Remarks: The GetMVVal function takes a long expression as its single argument. You must specify the number of the macro variable whose value you want to obtain. This value must be between 1 and 65535 for Macros\Basic 4.0 or higher and between 1 and 100 for Macros\Basic 3.0 or lower. Do not include the mv prefix. GetMVVal returns a string data type that contains the current value of the requested macro variable if the command was successful. If GetMVVal is unsuccessful, it returns an empty string value (““).

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether GetMVVal was successful. ErrOMNIC returns an integer value of zero if the Macros\Basic macro supplied the requested object value. If the request was not successful, ErrOMNIC returns the Visual Basic error code.

Example: This example uses GetMVVal to obtain the current value of macro variable mv3.

```
Dim lvVal As String

lvVal = GetMVVal(3)
If ErrOMNIC() = 0 Then
    MsgBox (“The current value of mv3 = “ & Format(lvVal, “0.00”))
Else
    ErrMsgBox
End If
```

GetOMNIC

This OmTalk routine returns the value of an OMNIC parameter.

Syntax: GetOMNIC(<Parameter name>)

Remarks: The GetOMNIC function takes a string expression as its single argument. You must specify the name of the OMNIC parameter whose value you want to obtain. The parameter name argument must contain a group name followed by a space and the parameter name.

GetOMNIC returns a String data type that contains the value of the requested parameter if the command was successful. If GetOMNIC is unsuccessful, it returns the text "#ERROR*n*#", where *n* is the Visual Basic error code, Err.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether GetOMNIC was successful. ErrOMNIC returns an integer value of zero if OMNIC supplied the requested parameter value. If the request was not successful, ErrOMNIC returns the Visual Basic error code.

Example: This example uses GetOMNIC to obtain the result of a noise calculation.

```
ExecuteOMNIC ("CalculateNoise")
If ErrOMNIC() = 0 Then
    noise$ = GetOMNIC("Result Current")
Else
    ErrMsgBox
End If
```

GetOMNICName

This OmTalk routine retrieves the full application name from OMNIC. Use it to determine which version of OMNIC or EZ OMNIC is currently running.

Syntax: GetOMNICName ()

Remarks: The GetOMNICName statement has no argument.

Example: This example uses GetOMNICName to retrieve the name of the current version of OMNIC.

```
dim szOmnicName as string
szOmnicName=GetOMNICName ()
```

GetOMNICVersion

This OmTalk routine returns the version number of OMNIC as a string.

Syntax: GetOMNICVersion (<avFormat>)

Remarks: The GetOMNICVersion statement has a single argument which can be used to control the format of the returned version number.

- avFormat = 0 - Returns the version number in native format with major and minor revisions separated by decimals. Example: 5.0.0.1
- avFormat = 1 - Returns the version number as a floating point number with minor revisions concatenated. Example: 5.001

Example: This example uses GetOMNICVersion to retrieve the version number in the native format.

```
dim lvVal as String
lvVal=GetOMNICVersion (0)
```

GetSpecCollectTime

This OmTalk routine returns a string that indicates the date and time when the currently selected spectrum was collected.

Syntax: GetSpecCollectTime(<format>)

Remarks: The argument <format> is an integer that determines the format of the returned date and time as follows.

- 1 - Returns a string containing a date and time in the format designated on the Date tab of the Regional Settings option in your computer's control panel.
- 2 - Returns a string containing the date and time in the same format as shown in the OMNIC spectrum collection and processing information.

Example: This example uses GetSpecCollectTime to display the time the selected spectrum was collected.

```
Dim lvTime As String

lvTime = GetSpecCollectTime(1)
MsgBox (lvTime)
```

GetSpecData

This OmTalk routine obtains numeric spectral data from an OMNIC spectrum and places it in an array.

Syntax: GetSpecData (<firstX>, <lastX>)

Remarks: The GetSpecData statement uses these arguments:

- firstX - A numeric expression that is the one boundary of a spectral region.
- lastX - A numeric expression that is the other boundary of a spectral region.

Both firstX and lastX should have the same unit as the X-axis.

GetSpecData returns the spectral data into the global array SpecData(). GetSpecData also sets the following global variables:

- GetSpecNum - The number of data points returned.
- GetSpecFirstX - The X-axis value of the first point in SpecData().
- GetSpecLastX - The X-axis value of the last point in SpecData().
- GetSpecIncrement - The data point spacing of SpecData() in X-axis units.

The variables above are declared as string data types. The SpecData() array is defined as a single data type.You do not need to define these variables in your code.

Use the function ErrOMNIC to test whether GetSpecData was successful. ErrOMNIC returns an integer value of zero if OMNIC supplied the requested spectral data. If the request was not successful, ErrOMNIC returns the Visual Basic error code.

(Continued on next page)

Example: The following example uses GetSpecData to obtain the spectral data from the region from 1620 to 1600 cm^{-1}. A message box displays the number of data points returned and the value of the first data point.

```
Dim tex as String
GetSpecData (1620, 1600)
if ErrOMNIC() = 0 Then
    tex = “Number of data points returned = “ + GetSpecNum()
    tex = tex + Chr(13) & Chr(10) & “1st point X value =”
    ↙& GetSpecFirstX()
    tex = tex & Chr(13) & Chr(10) & “1st point Y value =”
    ↙& SpecData(1)
    MsgBox (tex)
Else
    ErrMsgBox()
End If
```

GetVal

This OmTalk routine returns the numeric value corresponding to the word following the search string in a text string. This function has been made obsolete by use of the GetItem function in conjunction with the OMNIC parameter Result Array. See the Get Item function.

Syntax: GetVal(<text string> <search string>)

Remarks: The GetVal function uses these arguments:

- text string - The string expression being searched.
- search string - The string expression being sought.

The GetVal function always returns a Double data type. If the search string is not found in the text string, a value of zero is returned. GetVal ignores any ":" or "(" characters that may occur between the search string and the value that follows it.

The format for OMNIC text returned via the Result Current parameter is the same as that shown in the OMNIC readout or dialog boxes. For example, the CorrectedPeakArea command sets Result Current to "Area: 19.289 Uncorrected: 25.157 Region: (1468.346, 1423.154) Baseline: (1468.846, 1415.667)".

Example: This example uses GetVal to extract the values of the minimum and maximum from the result string of the OMNIC MinMax command. Note that the colon following the search string is optional.

```
Dim result as String
ExecuteOMNIC ("minmax")
ErrMsgBox()
result = GetOMNIC("Result Current")
ErrMsgBox()
min = GetVal(result, "Min:")
max = GetVal(result, "Max")
```

ItemCount

This OmTalk routine returns the number of items in a list as an integer data type.

Syntax: ItemCount(<list string>)

Remarks: The ItemCount function takes a string expression as its single argument. This expression is a list of separated values. ItemCount assumes items are separated by the list separator character specified on the Number tab of the Regional Settings application, located in the Windows Control Panel folder. ItemCount returns an integer data type that is the number of items in the list string. If the list string argument is empty, ItemCount returns a value of zero. This function (and also the GetItem and Pop functions) is useful when you want to obtain specific items from the list of values returned by the Result Array OMNIC parameter.

Example: This example displays a table of results from a Find Peaks operation.

```
  Dim lvList As String
  Dim lvTable As String
  Dim lvCount As Integer
  Dim lvIndex As Integer
  Dim lvPeakPosition As Single
  Dim lvPeakValue As Single

ExecuteOMNIC "PeakPick 0.25 50"
  If ErrOMNIC() = 0 Then
    lvList = GetOMNIC("Result Array")
    lvCount = ItemCount(lvList)
    For lvIndex = 5 To lvCount Step 2
      lvPeakPosition = GetItem(lvList, lvIndex)
      lvPeakValue = GetItem(lvList, lvIndex + 1)
      lvTable = lvTable & Format(lvPeakPosition, "0.00") & Chr(9)
      lvTable = lvTable & Format(lvPeakValue, "0.0000") & Chr(13)
      ↙& Chr(10)
    Next lvIndex
    MsgBox (lvTable, 64, "Find Peaks Result")
Else
  ErrMsgBox()
End If
```

LoadOmTalk

This OmTalk routine is used to load OmTalk.NET when starting an application.

Syntax: LoadOmTalk()

Example: This example loads the OmTalk.NET service so that it can be used.

```
Private Sub Form_Load(ByVal sender As System.Object, ByVal e As
↙System.EventArgs) Handles MyBase.Load
    LoadOmTalk()
End Sub
```

Pop

This OmTalk routine returns the first item in a list and removes this item from the list.

Syntax: Pop(<list string>)

Remarks: The Pop function takes a string expression as its single argument. This expression is a list of separated values. Pop assumes items are separated by the list separator character specified on the Number tab of the Regional Settings application, located in the Windows Control Panel folder.

Pop returns a string data type that is the value of the first item in the list string. If the list string argument is empty, Pop returns a null string value (““).

This function is useful when you want to process items from the list of values returned by the Result Array OMNIC parameter. See also the functions ItemCount and GetItem.

Example: This example stores the results of a Quantify operation in the array lvConc. Note how the array lvConc is dynamically allocated. This lets this code work for any number of results up to 40 components.

```
Dim lvQuantResult As String
Dim lvConc() As Single
Dim lvVal As String
Dim lvIndex As Integer

ExecuteOMNIC “Quantify”
lvQuantResult = GetOMNIC(“Result Array”)
If (ErrOMNIC() = 0) Then
    ReDim lvConc(40)
    lvIndex = 0
    lvVal = Pop(lvQuantResult)
    While (lvVal <> ““)
        lvIndex = lvIndex + 1
        lvConc(lvIndex) = lvVal
        lvVal = Pop(lvQuantResult)
    End While
    ReDim Preserve lvConc(lvIndex)
End If
```

SetApp

This OmTalk routine sets the value of a parameter or object property in a Windows application other than OMNIC.

Syntax: SetApp (<Application|Topic>, <Parameter>, <Value>)

Remarks: The SetApp statement uses these arguments:

- Application|Topic - A string expression that is the name of the application, the pipe char | (char code 124), and the topic you want to communicate with. Make sure there are no spaces in this string expression. This is the server or source application referred to in the Visual Basic documentation.
- Parameter - A string expression that is the name of the parameter or object property whose value you want to set.
- Value - A string expression that is the value you want to assign to the parameter or object property.

This statement opens a DDE conversation with the source application, sets the parameter or object property value, and then closes the DDE conversation.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether SetApp was successful. ErrOMNIC returns an integer value of zero if the application set the parameter or object property to the specified value. If SetApp was not successful, ErrOMNIC returns the appropriate Visual Basic error code.

Continue to use the SetOMNIC statement rather than this one if you are working with OMNIC. SetOMNIC is optimized to do a better job of managing the communication link with OMNIC.

Note The application must be running before the SetApp statement is executed. Unlike SetOMNIC, SetApp will not automatically start the application if it is not running. If the application is not running, the ErrOMNIC function will return the Visual Basic error code 282. ▲

(Continued on next page)

Example: This example sets the value of the cell in row two, column three to 0.153 in the Microsoft Excel spreadsheet with the name Sheet1. If Excel is not running, the Visual Basic Shell function is used to start Excel.

```
Dim lvHwnd As Integer

SetApp (“Excel|[Book1]Sheet1”, “R2C3”, “.153”)
If (ErrOMNIC() <> 0 Then)
    If ErrOMNIC() = 282 Then
        lvHwnd = Shell(“excel.exe”, 1)
        SetApp (“Excel|[Book1]Sheet1”, “R2C3”, “.153”)
    Else
        ErrMsgBox()
    End If
End If
```

SetMVVal

This OmTalk routine sets the value of a Macros\Basic macro variable.

Syntax: SetMVVal(<Macro variable number>, <Value>)

Remarks: The SetMVVal statement uses these arguments:

- Macro variable number - A long expression that is the number of the macro variable whose value you want to set. Do not include the mv prefix.
- Value - A string expression that is the value you want to assign to the macro variable.

The macro variable number must be between 1 and 65535 for Macros\Basic 4.0 or higher or between 1 and 100 for Macros\Basic 3.0 or lower.

It is not necessary for the macro variable to be declared in the Macros\Basic macro, since this routine will create the macro variable definition. This is useful when you want to pass values from your Visual Basic application back to a Macros\Basic macro.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether SetMVVal was successful. ErrOMNIC returns an integer value of zero if the Macros\Basic macro set the macro variable to the specified value. If SetMacro was not successful, ErrOMNIC returns the Visual Basic error code.

(Continued on next page)

Example: This example evaluates the first component value from a Quant result and then uses SetMVVal to set the value of macro variable mv3 to an appropriate text message.

```
Dim lvQuantResult As String
Dim lvText As String

ExecuteOMNIC "Quantify"
lvQuantResult = GetOMNIC("Result Array")
If ErrOMNIC() = 0 Then
    If .5 < Pop(lvQuantResult) < 1 Then
        lvText = "Product is within acceptable limits."
    Else
        lvText = "Product is out of spec."
    End If
    SetMVVal 3, lvText
End If
ErrMsgBox
```

SetOMNIC

This OmTalk routine sets an OMNIC parameter to a value that you specify.

Syntax: SetOMNIC (<Parameter name>, <Value>)

Remarks: The SetOMNIC statement uses these arguments:

- Parameter name - A string expression that is the name of an OMNIC parameter whose value you want to set. The parameter name argument must contain a group name followed by a space and the parameter name.
- Value - A string expression that is the value to set the parameter to.

Use the ErrMsgBox statement to display an error message that describes any error that may have occurred.

Use the function ErrOMNIC to test whether SetOMNIC was successful. ErrOMNIC returns an integer value of zero if OMNIC set the parameter to the specified value. If SetOMNIC was not successful, ErrOMNIC returns the Visual Basic error code.

Example: This Visual Basic.NET example uses SetOMNIC to set the spectral region before a noise calculation. It then obtains the result of the noise calculation.

```
Dim noise as String
SetOMNIC (“Display RegionStart”, 2250.0)
ErrMsgBox()
Dim lvEnd as Integer
lvEnd = 2200
SetOMNIC (“Display RegionEnd”, CStr(lvEnd))
ErrMsgBox()
ExecuteOMNIC (“CalculateNoise”)
If ErrOMNIC() = 0 Then
    noise = GetOMNIC(“Result Current”)
Else
    ErrMsgBox()
End If
```

Note If you are using Visual Basic 6.0, do not include parentheses around the parameters for SetOMNIC or ExecuteOMNIC. ▲

SetSpecData

This OmTalk routine sets the numeric values of spectral data in an OMNIC spectrum. For 32-bit Visual Basic projects, you must use this routine.

Syntax: SetSpecData

Remarks: The SetSpecData statement does not have any arguments.

SetSpecData sets the spectral data in the currently selected OMNIC spectrum to the values contained in the global array SpecData(). The following global variables specify the number of data points and region of data to be set:

- GetSpecNum - The number of data points to be set.
- GetSpecFirstX - The X-axis value of the first point in SpecData().
- GetSpecLastX - The X-axis value of the last point in SpecData().

The above variables and the SpecData() array are defined in the OmTalk Declarations section as single data types. You do not need to define these variables in your code.

Use the function ErrOMNIC to test whether SetSpecData was successful. ErrOMNIC returns an integer value of zero if OMNIC supplied the requested spectral data. If the request was not successful, ErrOMNIC returns the Visual Basic error code.

Example: This example uses SetSpecData to fit a straight line across the spectral data from the region 2400 to 2200 cm^{-1}.

```
GetSpecData (2400, 2200)
increment=(SpecData(GetSpecNum()) - SpecData(1)) / (GetSpecNum()-1))
For i = 2 To GetSpecNum()
    SpecData(i) = SpecData(1) + increment*(i-1)
Next i
SetSpecData()
```

StartOMNIC

This OmTalk routine runs OMNIC if it is not already running.

Note You do not need to call StartOMNIC before using the other OmTalk routines. If OMNIC is not already running when you try to communicate with it, it will be started automatically. Use StartOMNIC *only* if you must launch OMNIC with a specific window style or argument string. If OMNIC is already running when you call StartOMNIC, you may get an error message. ▲

Syntax: StartOMNIC(<Window style> <Argument string>)

Remarks: The StartOMNIC function has these arguments:

- window style - An integer corresponding to the style of the OMNIC window.
- argument string - A string expression containing any arguments or command line switches.

The following table identifies the possible values for the window style and the style of window that occurs as a result:

- 1 - The normal window with focus.
- 2 - Minimized with focus.
- 3 - Maximized with focus.
- 4 - The normal window without focus.
- 7 - Minimized without focus.

Set the argument string argument to an empty string if you do not use any command line options. Command line options are:

- <filename> - The name of a valid spectral data file that is to be displayed in the initial window.
- -b<benchfile> - Use benchfile instead of real bench.
- -i - Run in invisible mode; that is, with no user interface.
- -l <x,y> - The location of the upper-left corner, in pixels.
- -s<width,height> - The width and height, in pixels.

(Continued on next page)

If the StartOMNIC function is successful in starting OMNIC, it returns a Long data type task identification (ID) for OMNIC. The task ID is a unique number that identifies OMNIC. If StartOMNIC is unsuccessful in starting OMNIC, Visual Basic generates an error message.

Example: These examples use StartOMNIC to launch OMNIC. In the first example, the OMNIC window is minimized to an icon and the Visual Basic application retains the focus. In the second example, the OMNIC window is displayed in the upper-left corner of the screen, 820 wide by 520 high.

```
i = StartOMNIC(7, "")
i = StartOMNIC(1, "-10,0 -s820,520")
```

Strip

This OmTalk routine returns a string with any leading and trailing white space characters removed.

Syntax: Strip (<text string>)

Remarks: The Strip function takes a text string as its single argument. The function removes any carriage return, line feed, tab or space characters from the start and end of the string, and then returns the text string.

Example: This example removes white space characters from the start and end of a text string but leaves embedded white space alone.

```
Dim lvTex as String
Dim lvRetVal as String
lvTex = "  This is "&Chr(9) & "Column 1" & Chr(13) & Chr(10)
lvRetVal = Strip(lvTex)
```

UnloadOmTalk

This OmTalk routine is used to unload OmTalk.NET when you have finished using an application and are ready to close the application.

Syntax: UnloadOmTalk()

Example: This example will unload OMTalk.NET and, if necessary, make resources that are currently being used available for other applications.

```
Private Sub Form_Closed(ByVal eventSender As System.Object, ByVal
↙eventArgs As System.EventArgs) Handles MyBase.Closed
    UnloadOmTalk()
End Sub
```

The OMNIC DDE Application

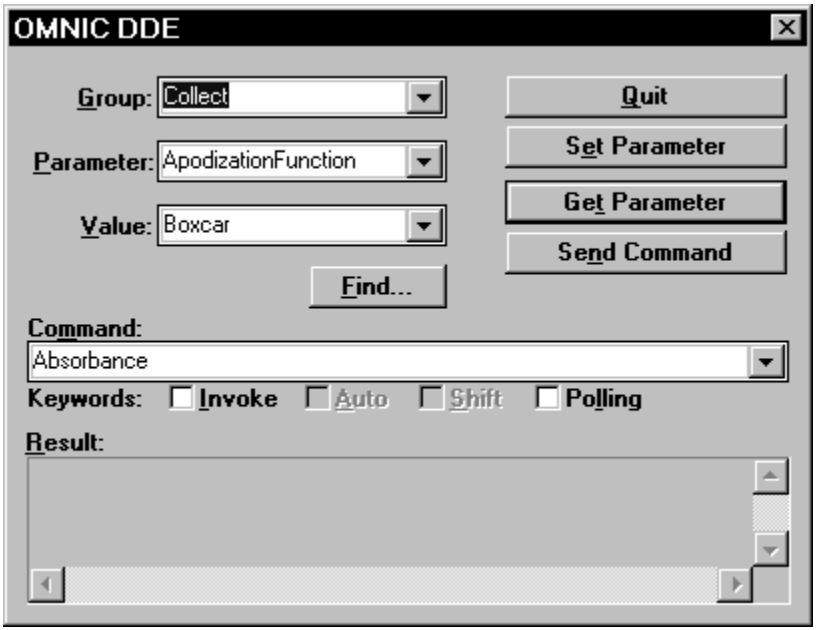
OMNIC DDE is a stand-alone application for sending commands to OMNIC and for setting and getting the values of OMNIC parameters via dynamic data exchange.

The application contains lists of OMNIC commands and parameters that are accessible via DDE. It also indicates the appropriate syntax and parameter values that may be used. A text search feature is provided which can assist you in locating a specific parameter or command in case you do not know the exact name or syntax.

This list may not include all of the commands and parameters described in the documentation. If you can not find the command or parameter you want, you can type it into the appropriate field.

General features

The OMNIC DDE application lets you test DDE commands and parameters. This is useful when you are writing or debugging Pro macros, because it shows you the exact syntax of the results.



The features of the OMNIC DDE application can be grouped into four categories:

- Parameters (group, parameter and value).
- Commands (command and keywords).
- Command buttons.
- Results.

Parameters (group, parameter and value)

Each OMNIC parameter belongs to a specific group. These groups are a way of organizing related parameters. For example, all data collection parameters are located in the Collect group. To access an OMNIC parameter, you must specify both the group name and the parameter name.

The Group drop-down list box displays the parameter groups recognized by OMNIC. To access the parameters in a specific group, choose a group from the Group list. The contents of the Parameter drop-down list box will change to display the parameters in the chosen group.

The Value drop-down list box displays the values permitted for the chosen parameter. Choose the appropriate value you want to use from this list. This box will be empty for many parameters; in this case, you may enter any value you wish. See the Macros\Pro on-line help for the type of value to use with each parameter.

The items listed in each drop-down list box are sorted alphabetically. You can choose an item by scrolling through the list or by typing the first few characters of an item while the list is displayed.

The Macros\Pro on-line help contains a description of every parameter available, but the drop-down lists may not include all of these. If you cannot find the parameter you want, you can type it into the appropriate field.

Commands (commands and keywords)

Each OMNIC command is listed in the Command drop-down list box along with any arguments or options that the command recognizes. The commands are sorted alphabetically. You can choose a command by scrolling through the list or by typing the first few characters of the command while the list is displayed.

The Macros\Pro on-line help contains a description of every command available, but the drop-down lists may not include all of these. If you cannot find the command you want, you can type it into the appropriate field.

Some commands require arguments. If arguments are required, a description of them is shown after the command name enclosed in angle brackets. Here is an example:

```
DeleteAnnotation <RegionStart> <RegionEnd>
```

The DeleteAnnotation command requires two arguments: RegionStart and RegionEnd. You must substitute values for these two arguments; these values must be separated by a space. An easy way to enter values for the arguments is to double-click the argument in the Command box and type the value. This will automatically substitute the value over the argument description place holder. For example, the following command deletes all annotation in the region from 4000 to 2000 cm^{-1} (assuming a wavenumber spectrum is currently selected).

```
DeleteAnnotation 4000 2000
```

Some commands have optional arguments. These arguments are enclosed in square brackets. Here is an example:

```
CoaddRegion [<StartTime> <EndTime> [<WindowTitle>]]
```

Optional arguments do not need to be provided. If you choose not to provide them, you must delete the optional argument place holders from the Command box. Do this by selecting all the text between, including the square brackets, and then pressing the Delete key.

In the previous example, there are two sets of optional arguments as indicated by the nested set of square brackets. This example can be interpreted as follows. If you provide a value for StartTime, you must also provide a value for EndTime because both of these arguments are within a set of square brackets. The WindowTitle argument is optional if you specify StartTime and EndTime. If you provide a value for WindowTitle, you must also provide values for StartTime and EndTime. You cannot specify a value for just the WindowTitle argument, because it is nested within the other option.

For example, the following commands are valid:

```
CoaddRegion

CoaddRegion 2.05 4.30

CoaddRegion 2.05 4.30 "Coadded Result"
```

The following command is not valid because no StartTime and EndTime values were provided:

```
CoaddRegion "Coadded Result"
```

Some commands have arguments that are not enclosed in angle brackets and are separated by a vertical bar, |. Here is an example:

```
OtherCorrections Dispersion|ATR
```

This notation means that you must choose one of the keywords separated by the vertical bar. You must delete the vertical bar and the keyword that you do not want to use.

Keywords are optional command arguments that affect the behavior of the command when it is executed. To add a keyword to the command, click the appropriate check box after choosing a command from the Command drop-down list box.

The available kinds of keywords are explained below.

Invoke lets you specify interactive operation for a command. When used with commands like CollectSample or Subtract, the interactive versions of these commands are "invoked." For example, the Collect Sample window appears during data collection, and the Subtract window appears allowing you to perform an interactive subtraction. When used with commands like Average, the result will be displayed in a dialog box even though there is no equivalent OMNIC menu command.

When the Invoke keyword is used with a command, execution of code pauses until the operator closes the interactive window or dialog box. In other words, the DDE conversation that initiates the command is not completed until the window or dialog box is closed.

The Invoke keyword also affects how error messages are handled. If a command is used without this keyword, errors are stored and must be retrieved by getting the value of the parameter Result Error. When the Invoke keyword is used, these errors are displayed to the operator and a response is required.

Auto sets up data collection so that no operator prompts for entering a title and preparing for data collection are displayed. This check box is enabled only for data collection commands that allow this option. The Invoke keyword must always be selected when the Auto keyword is selected.

Shift is used with the Select command to cause the specified spectrum to be selected in addition to the currently selected spectrum or spectra. This keyword is also used with the PeakHeight command to seek the peak closest to the specified peak location.

Polling causes OMNIC to complete, or close, the DDE conversation as soon as the command is initiated. Without this keyword, OMNIC holds on to the DDE conversation until the command has finished executing, and then closes the conversation.

This keyword may be used with data collection commands to initiate a data collection and then immediately return. This allows your program to continue running while OMNIC proceeds with data collection.

This keyword gets its name from the polling mechanism you use in your code to test to see if the command has completed. Test the appropriate MenuStatus group parameter to see if its value is Enabled or Disabled.

For example, the MenuStatus CollectSample parameter remains Disabled until data collection has finished, then its value becomes Enabled.

Result contains the results of the Set Parameter, Get Parameter and Send Command buttons.

For the Get Parameter button, this field contains the current value of the designated OMNIC parameter if the command is successful. If the command fails, an error message is displayed in this field. The format of the result is exactly the same as that which would be obtained via the GetOMNIC function in OmTalk. You may find this useful for experimenting with parameters so you can see how to deal with their results in your Macros\Pro code. Note that the Result field is editable; you may cut or copy any text in this field to the Clipboard.

For the Set Parameter and Send Command buttons, the result will be "OK" if the operation was successful, or an error message if the operation was unsuccessful.

The error or failure messages are somewhat generic, something like "Foreign application won't perform DDE method or operation." The most prevalent reasons for failure are the following:

- Improper syntax in the Value drop-down list box when using the Set Parameter button.

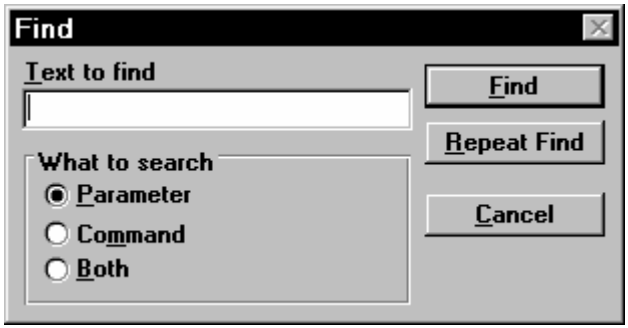
- Failure to remove or substitute valid values for argument or option place holders in the Command drop-down list box.

Refer to the Macros\Pro on-line help system to see what values OMNIC recognizes for each parameter and command.

Command buttons

These are the buttons in the application that carry out a specific action.

Find displays a dialog box that assists you in locating a specific command or parameter. Use this feature when you don't know the exact name of a command or parameter or the group to which a parameter belongs.



For example, suppose you want to find the parameter that returns error information. Type "error" in the Text To Find box, select Parameter in the What To Search box and click the Find button. The application searches all parameter names until it finds one containing "error". The first occurrence is the parameter Collect CorrError. This parameter is displayed in the Group and Parameter boxes with the matching text highlighted. To continue the search, click the Repeat Find button. The search continues and finds the next parameter, Result Error. This is the parameter we want, so you can click the Cancel button in the Find dialog box to close it.

Use the Find button to repeat the search of all parameters or commands; use Repeat Find to continue the search after the last occurrence.

The search is not case sensitive; this means "error" will match both "error" and "Error".

Other functions

The OMNIC DDE application also uses these buttons.

Quit exits the OMNIC DDE application.

Set Parameter sets an OMNIC parameter to the value specified in the Value drop-down list box. If the operation is successful, "OK" is displayed in the Result field. If the operation fails, the reason is displayed in the field.

Get Parameter obtains a parameter's value from OMNIC after you have selected a parameter. The current value of the selected parameter is displayed in the Result field.

Send Command executes the command exactly as it appears in the Command drop-down list box along with the selected keywords. You must have already substituted values for any argument or option place holders in the command string. The result of the command is displayed in the Result field.

Messages

This section explains the various messages you may encounter when using OMNIC DDE.

- "No match found."

  This message is displayed when you click the Find button in the Find dialog box, and there are no parameters or commands that contain the text you have entered in the Text To Find box.

- "No additional matches found."

  This message appears when you click the Repeat Find button in the Find dialog box, and there are no additional parameters or commands that contain the text you have entered in the Text To Find box. When you click OK to close this message, the last parameter or command that was found is displayed in the Parameter or Command drop-down list box.

- "Executing command..."

  This message is displayed in the Result field while OMNIC is executing a command, after you click the Send Command button while in stand-alone mode. As soon as the command finishes and the DDE conversation is closed, this message is overwritten with the result of the command.

- "Setting parameter value..."

  This message appears in the Result field while OMNIC is setting the value of a parameter, after you click the Set Parameter button while in stand-alone mode. As soon as the DDE conversation is closed, this message is overwritten with the result of the Set Parameter operation.

- “Reading parameter value...”

  This message is displayed in the Result field while OMNIC is getting the value of a parameter, after you click the Get Parameter button while in stand-alone mode. As soon as the DDE conversation is closed, this message is overwritten with the value of the parameter.

- “No Macros\Pro software on this system. You may not use this routine.”

  This message appears if a licensed copy of Macros\Pro software cannot be found on your system.

OMNIC Commands and Parameters

You can use the OMNIC commands and parameters within the Visual Basic projects you create to automate OMNIC software operations. The OMNIC command language provides all of the OMNIC software commands and parameters plus commands and parameters for performing additional operations.

Note Versions of OMNIC earlier than OMNIC 6.0 may not support all of the current commands and parameters. ▲

Note Complete descriptions of the DDE commands and parameters are available in the on-line Macros/Pro help system. ▲

The following list provides some general information about the OMNIC command interface.

- The language is not case sensitive.
- Command arguments are separated by spaces. If an argument includes embedded spaces, the argument must be enclosed in double quotation marks.
- Setting parameters causes them to take effect immediately. However, it is illegal to set bench or collect parameters while data collection is in progress.
- The Invoke keyword may be used with any OMNIC command that displays a window or dialog. It takes another OMNIC command as its first argument. When the Invoke keyword prefaces a command, the interactive form of the command is invoked. When the macro is run, the macro pauses until the operator closes the window or dialog box. For more information, refer to the Invoke entry in the OMNIC commands section.

- If you want to pass long filenames that contain spaces to OMNIC, enclose the filenames in double quotation marks.

- An invisible window is created when OMNIC is started. The main purpose of this window is to hold spectra that haven't been placed in a visible window. For example, when a new spectrum is collected, the result is a new spectrum that will belong to the invisible window. This spectrum can then be displayed in a visible window with the Display command. Alternatively, it can be operated on in the window without ever appearing on the screen if no OMNIC windows are open. For example, a sample can be collected and the height of a peak calculated without ever putting the sample in a visible window. The title of this window is InvisibleDDEWindow.

Syntax rules

If you are using the OMNIC commands and parameters with OmTalk and Visual Basic, the following syntax rules apply:

- OMNIC commands are executed through the ExecuteOMNIC statement.
- The command string, including all arguments, that follows the ExecuteOMNIC statement must be enclosed in double quotation marks. For example, to issue a CollectSample command and give the spectrum the title “Polystyrene,” you would type:

  ```
  ExecuteOMNIC (“CollectSample Polystyrene”)
  ```

  If the arguments include embedded spaces, the argument must be enclosed in two sets of double quotation marks in addition to the entire command string being enclosed in double quotation marks. For example, to issue a CollectSample command and give the spectrum the title “This is Sample1,” you would type:

  ```
  ExecuteOMNIC (“CollectSample ““This is Sample1””””)
  ```

- Parameters are set with the SetOMNIC statement and retrieved with the GetOMNIC function.

- The parameter string that follows the SetOMNIC statement or the GetOMNIC function must be enclosed in double quotation marks. For example, to set the Display XStart parameter to 4000 cm^{-1}, you would type:

  ```
  SetOMNIC (“Display XStart”, 4000)
  ```

  To get the XStart value, you would type:

  ```
  XStart = GetOMNIC (“Display XStart”)
  ```

- If you perform an operation that produces a text result, such as calculating the noise in a region, the result is obtained by using the GetOMNIC function to retrieve the parameter Result Current. The parameter Result Array may also be used to obtain numerical values without the full text of the result. The items in Result Array correspond to the numerical results in Result Current. There is also a Result Error that can be read to get the last error that occurred.

If you are using the OMNIC commands and parameters with applications other than OmTalk and Visual Basic, the following syntax rules apply:

- Commands must be enclosed in square brackets.
- Multiple commands can be passed in one message, separated by semicolons (i.e., [command1;command2;command3]).

For example, Macro1 is a Microsoft Word macro that opens a spectrum file, calculates noise between 2300 - 2000 cm^{-1}, then inserts the result into a Word document. Macro2 is a Microsoft Excel macro that opens a spectrum file, calculates the height of the peak closest to 1600 cm^{-1}, then inserts the resulting peak location and height into an Excel spreadsheet.

```
Sub Macro1()
  'Example Word macro.
  chan = DDEInitiate(App:="OMNIC", Topic:="Spectra")
  DDEExecute Channel:=chan, Command:="[Import_
  ↙""c:\omnic\spectra\absorb.spa""]"
  DDEPoke Channel:=chan, Item:="Display RegionStart",
  ↙Data:="2000"
  DDEPoke Channel:=chan, Item:="Display RegionEnd",
  ↙Data:="2300"
  DDEExecute Channel:=chan, Command:="[CalculateNoise]"
  returnValue = DDERequest(Channel:=chan, Item:="Result
  ↙Current")
  DDETerminate Channel:=chan
  ActiveDocument.Content.InsertAfter Text:=returnValue
End Sub
```

```
Sub Macro2()
   'Example Excel macro.
   channelNumber = Application.DDEInitiate( _
   app:="OMNIC", _
   topic:="Spectra")
  Application.DDEExecute channelNumber,_
  ↙"[Import ""c:\omnic\spectra\absorb.spa""]"
  Application.DDEExecute channelNumber,_
  ↙"[PeakHeight 1600 Shift]"
  returnValue = Application.DDERequest(channelNumber,_
  ↙"Result Array")
  Application.DDETerminate channelNumber
  Worksheets("Sheet1").Range("A1").Value = returnValue
End Sub
```

Note There is a set of Atlμs™ commands that you use to create Pro macros for the Atlμs application. For information about these commands, including a complete list with descriptions, please see the Macros\Pro on-line help. ▲

Bench and Collect parameters for step-scan experiments

The Spectral Resolution, Points Before Peak, and Sample Spacing parameters that appear in the Amplitude Modulation and Phase Modulation setup screens correspond to the following OMNIC Bench and Collect group parameters.

Setup Parameter	Parameter	Notes
Sample Spacing	Bench SSP	1.0, 2.0, or 4.0 allowed
Spectral Resolution	Collect Resolution	0.125 - 32.0 cm^{-1} allowed
Points Before Peak	Collect PeakPosition	must be no greater than 32768/ (Resolution*SSP)

Other Bench group parameters that apply to step-scan experiments in the usual way are: ADC, Aperture, BeamPath, BeamSplitter, Gain, HighCutoff (limited by SSP), LowCutoff, and Source.

Other Collect group parameters usable in step-scan experiments are: ApodizationFunction, AutoSave, BackgroundFileName, BackgroundHandling (AfterTime and ThisBkg options only), BaseName, BasePathName, DataCorrections, FinalFormat, MaxBackgroundAge, NumPhaseDataPts (limited by PeakPosition), NumPhaseTransformPts, PhaseCor, SequenceNum, and ZeroFill.

The following Bench group parameters are not used during step-scan collects: BidirectionalScan, RapidScanState, and Velocity.

The following Collect group parameters are not used during step-scan collects: Autogain (always False), Correlation, CorrError, ExternalTrigger (always False), NumDataPts (calculated from Resolution and SSP), NumScans (always 1), NumTransformPts (calculated from NumDataPts and ZeroFill), and SaveInterferograms (always True).

Macros\Pro Examples

We have provided you with a set of example macros on the Macros\Pro software disk. These macros were installed in the My Documents\OMNIC\PROMACS\EXAMPLE\VB# directory when you installed the Macros\Pro software (where VB# represents your version of Visual Basic). Each example is then stored in its own directory.

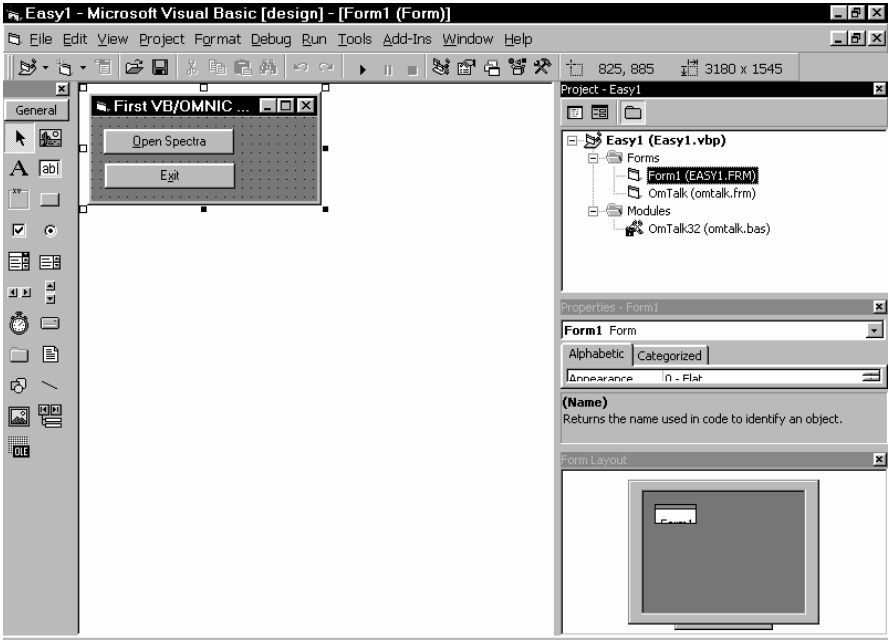
These macros can be opened in Visual Basic if you want to view them or if you want to copy code from the examples into your own macros. Depending on the version of Visual Basic you are using, you may get this message: "This file was saved in a previous version of Visual Basic." Click OK to save in the current Visual Basic format.

This chapter provides brief descriptions of the contents and purpose of each example macro provided with your Macros\Pro software. For a detailed introduction to creating Pro macros using Visual Basic, see the "Creating Pro Macros With Visual Basic" chapter.

Note Some of the examples in this chapter use OMTALK.BAS and OMTALK.FRM and are not compatible with Visual Basic.NET. A Visual Basic.NET example can be found in the My Documents\OMNIC\PROMACS\EXAMPLE\VBNET directory. ▲

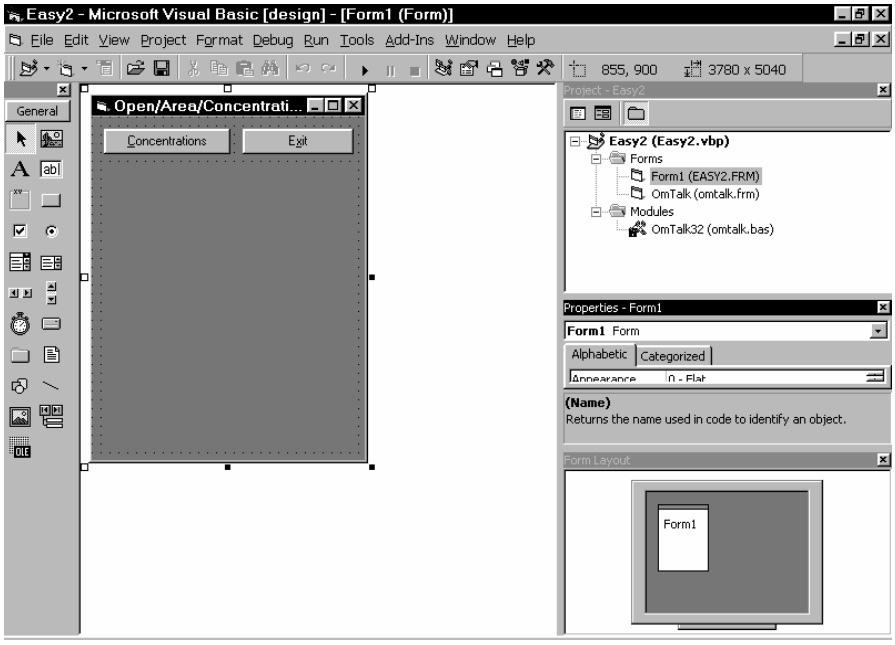
Visual Basic example 1: EASY1.VBP

This is a simple Visual Basic project that opens two of the example spectra that are installed as part of the standard OMNIC FT-IR software and puts them into a spectral window that is set up as a two-pane stack. The project includes the OMTALK files and a single user form containing two buttons.



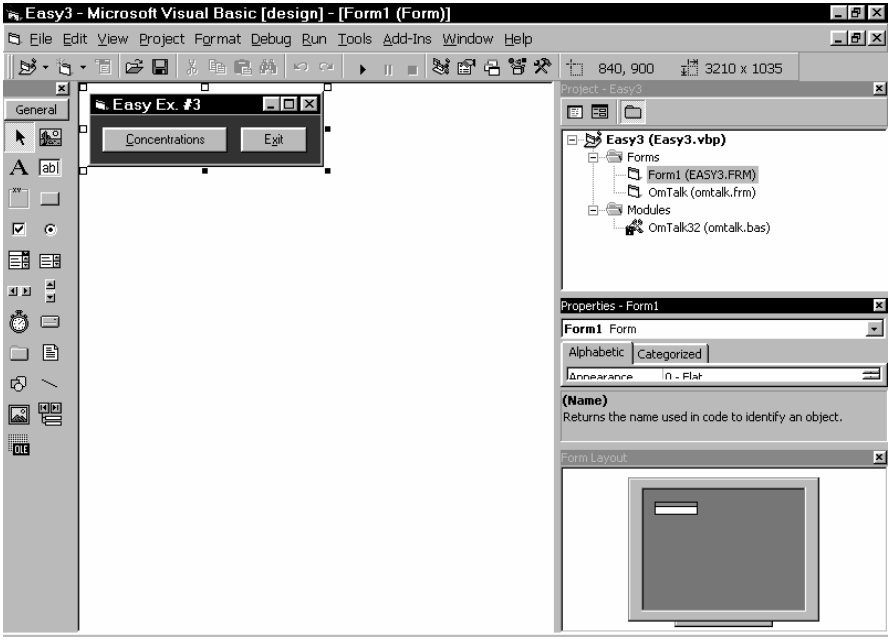
Visual Basic example 2: EASY2.VBP

This Visual Basic project sequentially opens five spectra, calculates a peak area for each spectrum and then uses this area and a calibration equation to determine the concentration of a component. The example spectra are automatically installed with the Macros\Pro software. The project consists of the two OmTalk files and a single user form containing two buttons and a text field.



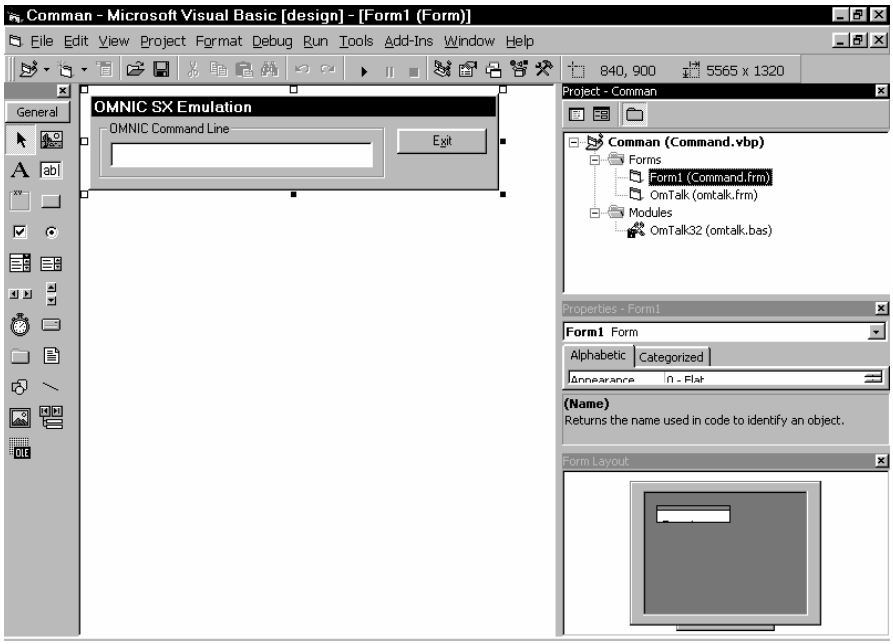
Visual Basic example 3: EASY3.VBP

This Visual Basic project demonstrates how the results obtained from OMNIC through a Visual Basic program can be sent via DDE to another program, in this case Microsoft Excel. For this program to run properly, you must have Excel installed on your system in your DOS path. The project also assumes that you do not have any files in the XLSTART directory that are launched automatically when Excel is started. The same area and concentrations are calculated as in the example project EASY2.VBP, but in this case the results are sent to the Excel spreadsheet.



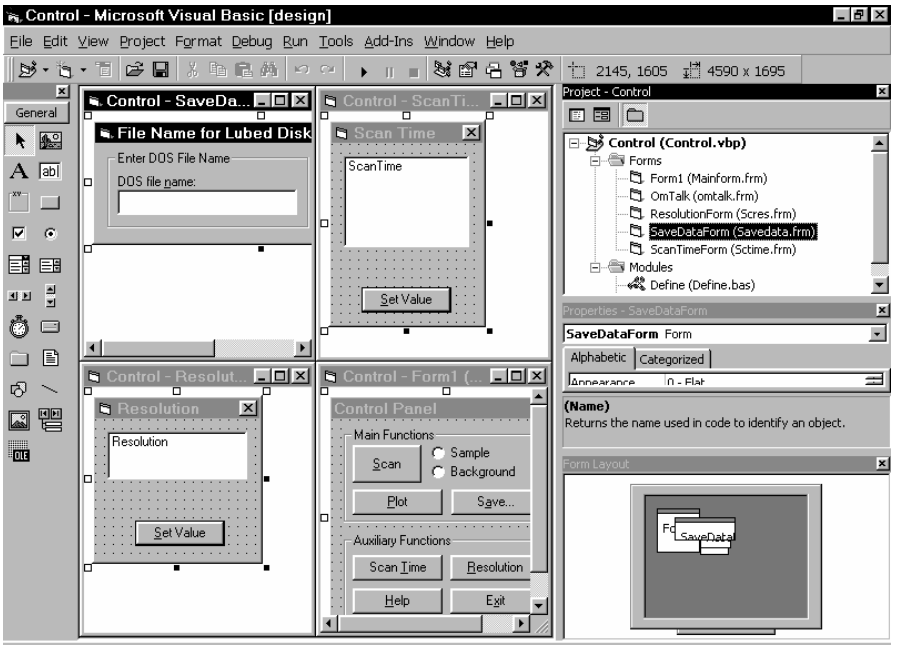
Visual Basic example 4: COMMAND.VBP

This Visual Basic project demonstrates how text can be entered and interpreted as commands and parameters that can then be sent to OMNIC. A text line is created and the user can enter multiple commands and parameters, followed by the Enter key. A function call then breaks the string into text strings that are interpreted and sent to OMNIC. The project, located in Command directory, consists of the OMTALK files and a single user form that contains the command entry text field and an Exit button. The command line allows multiple commands and parameters to be entered on a single line separated by spaces. Filenames must include extensions.



Visual Basic example 5: CONTROL.VBP

This Visual Basic project demonstrates how a special OMNIC remote control operation panel can be developed. This type of project can be used to create simple interfaces for operators. Besides the normal OMTALK files, this program consists of four forms and a global definition section. MAINFORM.FRM is the main control form that has the OMNIC operation buttons. SCTIME.FRM contains the choices for scan time, and SCANRES.FRM contains the choices for resolution settings. SAVEDATA.FRM appears when the file is to be saved on the disk. The data collection uses the Auto command suffix in the OMNIC collection procedure, but the comments explain how a custom collect, can be substituted.



Features you should note in this macro include the following:

- Use of Option Explicit in form declaration. This option forces you to declare the type of all variables using the Dim statement. This is good programming practice because it helps trap potential bugs.

- Center form code in form_load procedure.

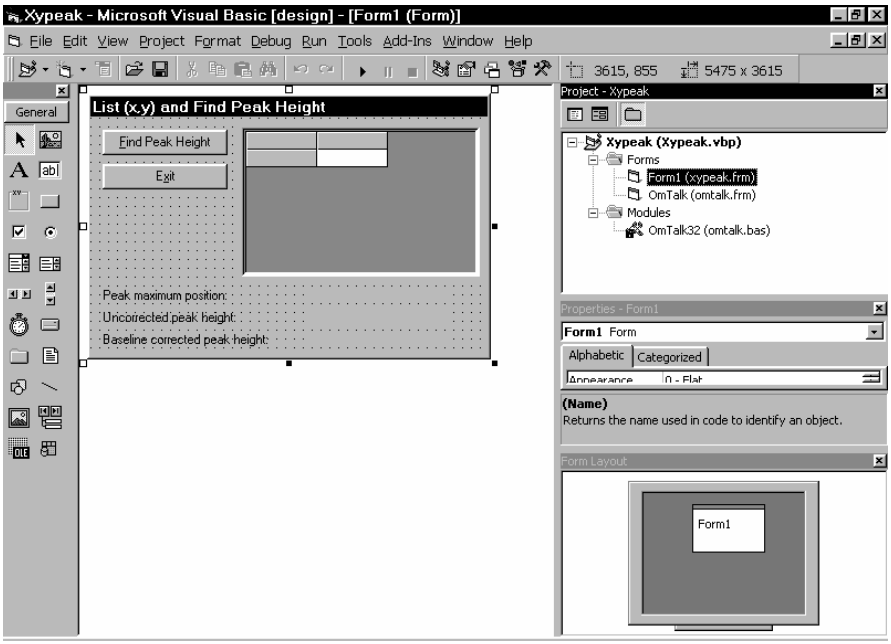
- Use of ErrOMNIC() and ErrMsgBox() procedures to check for successful completion of OMNIC actions.

- Use of a procedure (SetValue) called by several actions (button click and list box double-click). Note that this procedure is declared as private, meaning it is available only to the form in which it resides. Both the Resolution and Scan Time forms contain a procedure named SetValue. This procedure is different for each form and can be called only by other procedures in the same form.

- Use of module (define.bas) to declare global variables and Windows API (Application Programming Interface) functions. These variables and procedures can be used by any form in the project.

Visual Basic example 6: XYPEAK.VBP

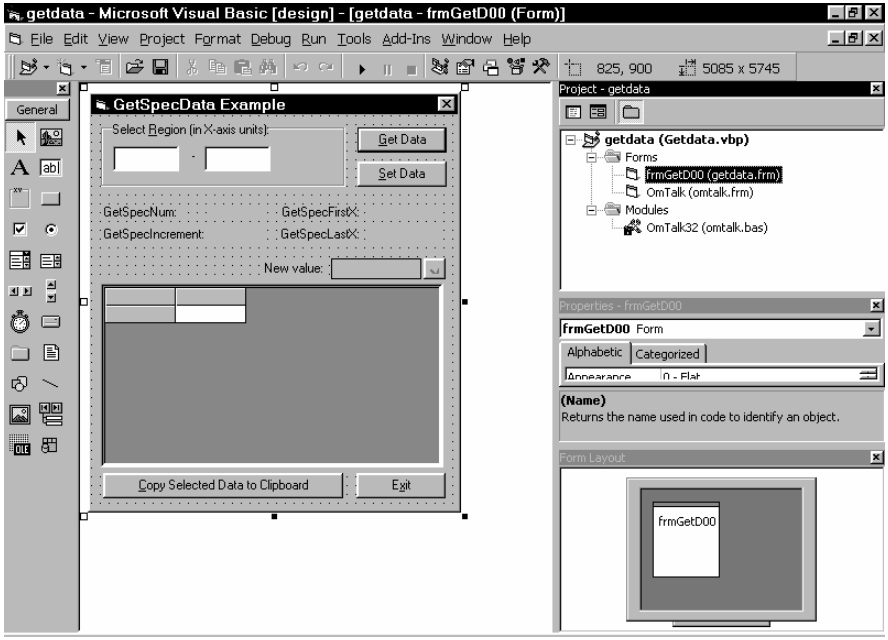
This Visual Basic project demonstrates how an OMNIC spectrum can be loaded automatically and saved as an X,Y pair file. This file is then opened into Visual Basic arrays for manipulation. In this example, a corrected peak height is calculated for the polystyrene band around 1605 cm^{-1}. The average absorbance in the baseline region is calculated on both sides of the peak and used with the midpoints of the baseline region (cm^{-1}) positions to calculate the equation of the line through the two points. The arrays are then evaluated to find the peak maximum, and the maximum height is corrected based on the equation developed using the baseline points. In addition to the two OmTalk files there is one form with two buttons, a spreadsheet and three text boxes.



This macro uses the grid32.ocx Active-X control.

Visual Basic example 7: GetData.VBP

This Visual Basic project demonstrates how data can be read directly from the selected OMNIC spectrum via DDE. The range of data to be read is defined using the OMNIC region tool. The data is obtained using the OmTalk GetDataArray subroutine, which is loaded into a Visual Basic spreadsheet and can be placed onto the Clipboard. This project consists of the OmTalk files and a single user form that contains four buttons, a spreadsheet and six text boxes.

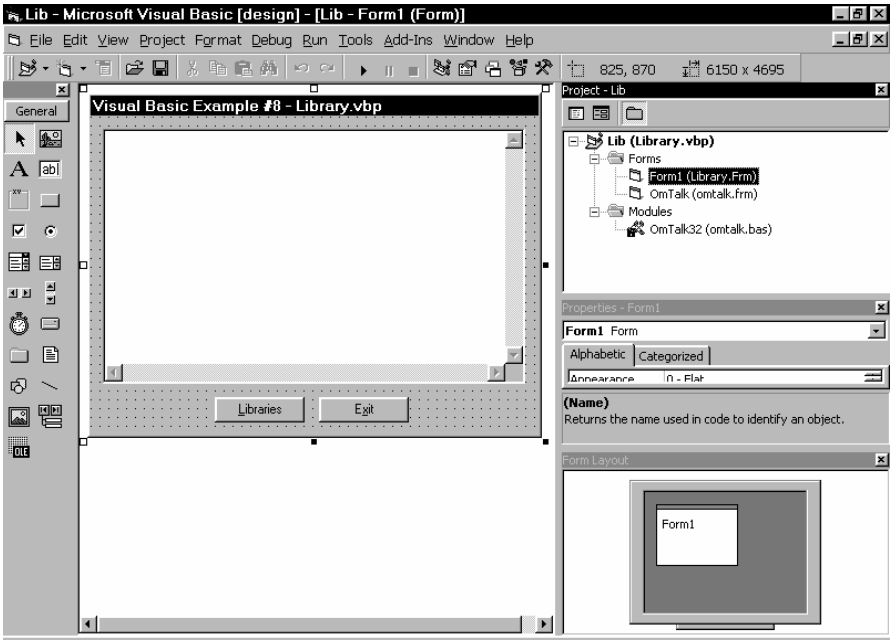


Features you should note in this macro:

- Uses the grid32.ocx Active-X control.
- Uses GetSpecData and SetSpecData procedures instead of GetDataArray and SetDataArray. (GetDataArray and SetDataArray can be used with 16-bit applications only. You must use the GetSpecData and SetSpecData for 32-bit applications.)

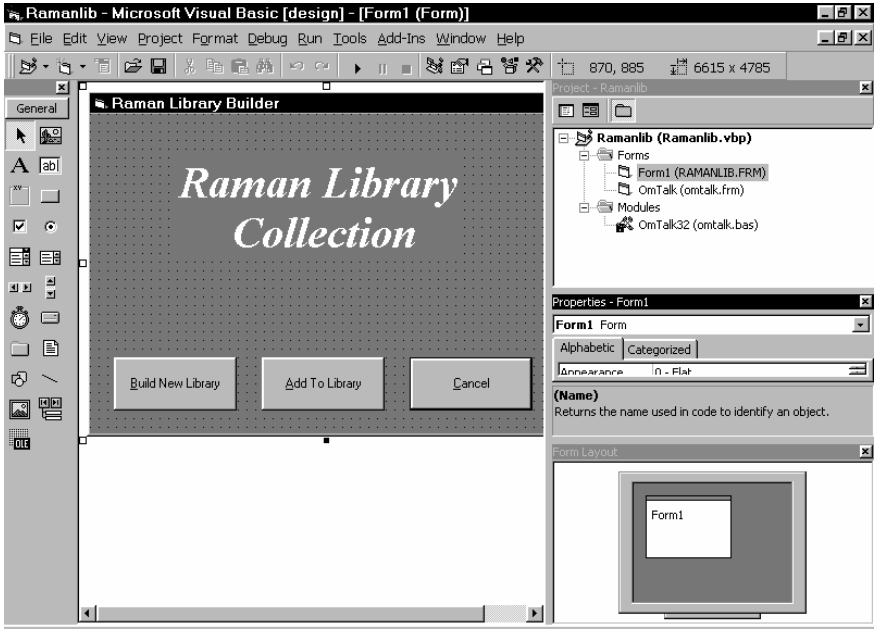
Visual Basic example 8: LIBRARY.VBP

This Visual Basic project details the use of a number of the special library commands and parameters. The libraries are stored on the disk as numbered files and referred to in OMNIC dialog boxes by descriptive titles. This project, in the Library directory, creates a list that relates the descriptive titles and the numerical filenames. In addition to the two OMTALK files there is one form with two buttons and a scrolling text box.



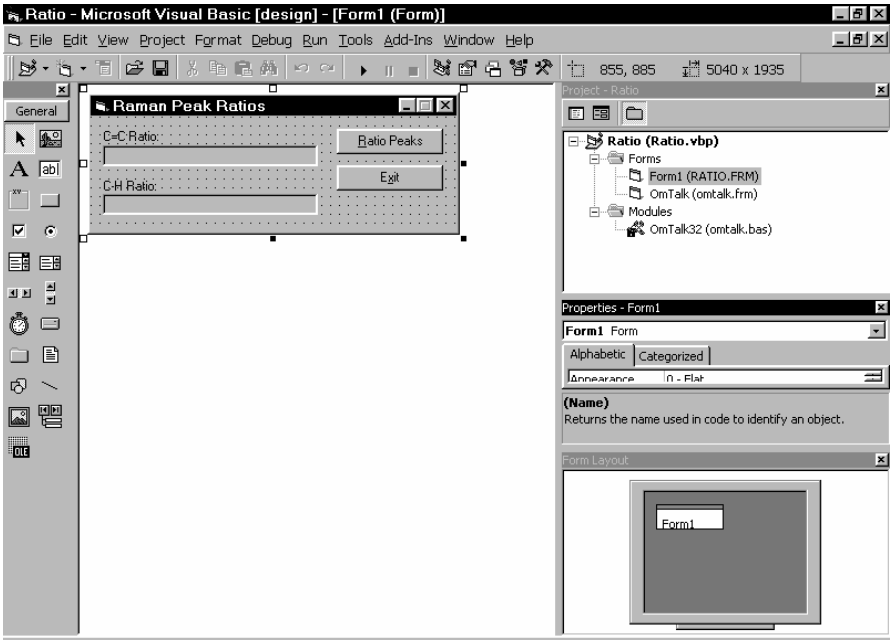
Visual Basic example 9: RAMANLIB.VBP

This Visual Basic project simplifies the building of custom Raman libraries. Samples can be collected and added to create a new Raman library or to expand an existing Raman library. In addition to the two OMTALK files there is a single user form containing three buttons. This example is in the Ramanlib directory.



Visual Basic example 10: RATIO.VBP

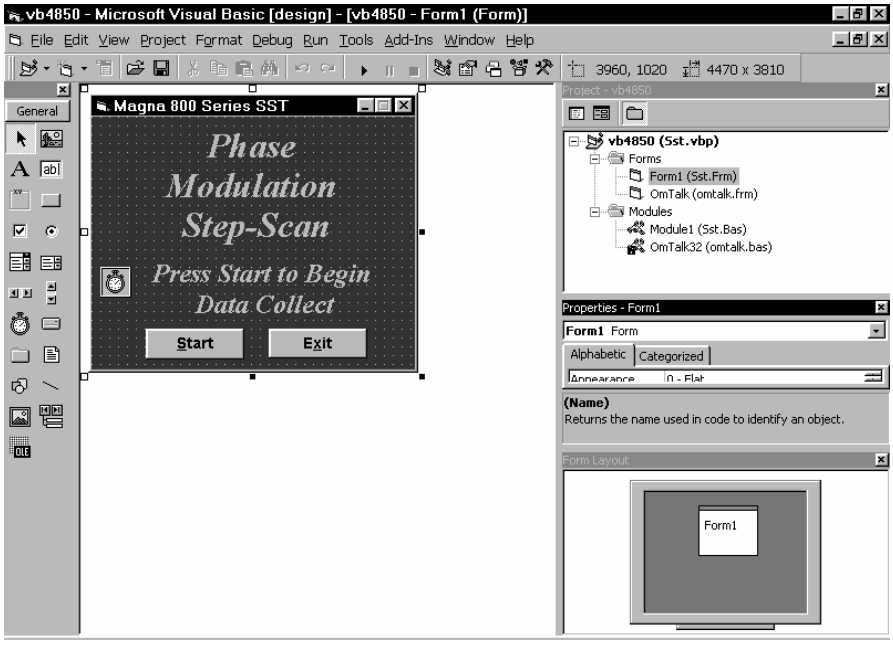
This Visual Basic project opens a Raman spectrum and calculates three corrected peak heights. Two of the peak heights are then ratioed to the third peak height. The ratios are put into text boxes and can be compared directly or put into a calibrated routine to determine what concentration is present. In addition to the two OMTALK files there is one form with two buttons and two text boxes.



This macro demonstrates how to tell Raman spectra from other types using the RamanLaserFreq parameter.

Visual Basic example 11: SST.VBP

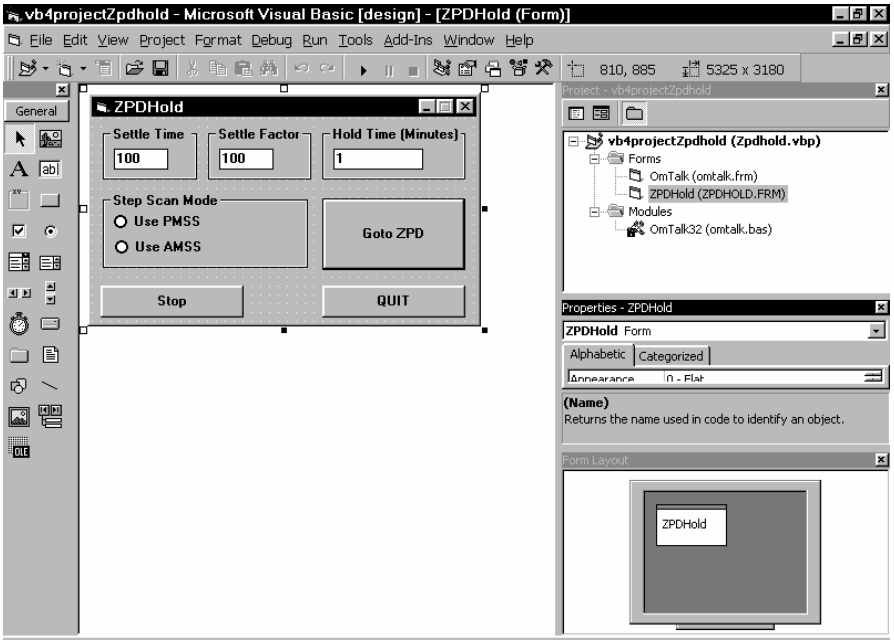
This Visual Basic project includes Nexus® 870 (or Magna-IR® 850 or 860) examples using OMTALK subroutines. Several phase modulation step-scan spectral collections are performed at various modulation frequencies. In addition to the two OMTALK files, there is one form containing two buttons. This example is in the SST directory.



Code in the macro traps for the case when this example is run on a system that does not have SST™ software installed.

Visual Basic example 12: ZPDHOLD.VBP

This Visual Basic project includes Nexus 870 (or Magna-IR 850 or 860) examples using OmTalk subroutines. The moving mirror is positioned at the interferogram peak location (ZPD). This facilitates the alignment of external optics and adjustment of external electronics for optimum performance during a step-scan data collection. Spectra collected with this macro will exhibit strange artifacts due to the parameters used to engage an extended hold time at the interferogram peak. Perform a collection with proper experimental parameters for valid spectra. In addition to the two OmTalk files, there is one form containing three buttons, three text boxes and two radio buttons.



Using OMNIC QuantPad™ DDE commands and parameters

This section describes how to use some of the commands and parameters that have interactions. Not all commands and parameters are shown, only those requiring further explanation.

Accessing report information

The information for the report is stored in an internal table in the GSANAL parameter set. The individual elements of the table can be accessed through the use of several commands and parameters.

Reading information from the table

1. Initialize the current set of information in the GSANAL parameter set with the command:

   ```
   FirstReportComp
   ```

2. Remove the first set of information from the table by reading the following parameters.

   ```
   ReportCompUse
   ```

3. Step to the next set of information in the table with the command:

   ```
   NextReportComp
   ```

4. Repeat steps 2 and 3 for NumMethodComp times (A parameter in GSANAL parameter set).

Updating the table with new information

1. If an element of the table needs to be changed, set the GSANAL parameter CurReportComp to the element number in the list that is to be changed.

2. Write the information to all of the parameters in the list. It is important to update all of them at the same time.

   ```
   ReportCompUse
   ```

3. Save the new information into the table with the command:

   ```
   UpdateReportComp
   ```

Using Commands and Parameters With Other Applications

Pro macros use the Dynamic Data Exchange (DDE) capabilities of Windows and OMNIC to automate OMNIC software operations. This manual has provided information on developing Pro macros using Visual Basic and has described the use of the OmTalk routines for handling the DDE interactions between Visual Basic and OMNIC.

The OMNIC commands and parameters can be used with other programs that support DDE. However, OmTalk can be used only with Visual Basic. Therefore, all of the DDE interactions between the programming language you choose and the OMNIC commands and parameters must be handled using the programming language. The following section provides a general introduction to DDE. For detailed information on DDE, refer to the manual for the programming language that you have chosen.

Dynamic data exchange basics

This section describes the use of some of the commands and parameters that have interactions. Not all commands and parameters are shown, only those requiring further explanation.

Dynamic Data Exchange is defined as the form of interapplication communications used by Microsoft Windows programs to support the exchange of commands and parameters between applications. This communication takes the form of a conversation that is similar to the conversation between two people. A DDE conversation establishes a temporary or permanent link between two Windows applications. This link acts as a conduit for the exchange of information between the connected applications. The exchanged data can be information that is copied from one application to the other, or commands for the other application to process.

In a DDE conversation the application that initiates the conversation is known as the *destination application*. or simply the *destination*. The application responding to the conversation is called the *source application* This terminology may seem backward, but keep in mind that the application that initiates the conversation usually wants some information to be sent to it (*destination* of information) by the responding application (*source* of information). An application may be involved in several conversations at the same time.

To initiate a DDE conversation, the destination application sends a message to Windows defining a source application that it wants to communicate with and a topic for the conversation. The topic defines the subject of the conversation and usually relates to some unit of source application data. For OMNIC the topic is always Spectra.

Windows applications that support DDE are always listening for conversations that refer to them. When a source application receives a request to have a conversation concerning a topic that it recognizes, it responds by starting a conversation. Once the conversation starts, the topic cannot be changed unless the conversation is ended and a new one is initiated. During the conversation the source and destination applications can exchange information concerning *items* in a bi-directional manner. *Items* consist of data or commands that are meaningful to both the source and destination applications. The item can be changed by either the source or destination during any given conversation.

There are many Windows compatible programming environments that can be used with OMNIC via DDE. High level Windows compatible languages such as Borland Turbo Pascal® for Windows and Microsoft C can be used to create advanced macros that will interact with OMNIC. You can also use the OMNIC commands with the SmartPad® software from Softblox, Inc., that is included with the OMNIC Utilities software.

Syntax rules for DDE conversations

If you are using the OMNIC commands and parameters with other applications, the following syntax rules apply:

- In DDE conversations, you must specify the name of the application and the topic of the conversation. The application name for OMNIC is “OMNIC”; the topic is “SPECTRA”.
- Commands must be enclosed in square brackets.
- Multiple commands can be passed in one message, separated by semicolons (i.e., [command1;command2;command3]).

For example, Macro1 is a Microsoft Word macro that opens a spectrum file, calculates noise between 2300 - 2000 cm^{-1}, then inserts the result into a Word document. Macro2 is a Microsoft Excel macro that opens a spectrum file, calculates the height of the peak closest to 1600 cm^{-1}, then inserts the resulting peak location and height into an Excel spreadsheet.

```
Sub Macro1()
  'Example Word macro.
  chan = DDEInitiate(App:="OMNIC", Topic:="Spectra")
  DDEExecute Channel:=chan, Command:="[Import_
  ↙""c:\omnic\spectra\absorb.spa""]"
  DDEPoke Channel:=chan, Item:="Display RegionStart",
  ↙Data:="2000"
  DDEPoke Channel:=chan, Item:="Display RegionEnd",
  ↙Data:="2300"
  DDEExecute Channel:=chan, Command:="[CalculateNoise]"
  returnValue = DDERequest(Channel:=chan, Item:="Result
  ↙Current")
  DDETerminate Channel:=chan
  ActiveDocument.Content.InsertAfter Text:=returnValue
End Sub
```

```
Sub Macro2()
   'Example Excel macro.
   channelNumber = Application.DDEInitiate( _
   app:="OMNIC", _
   topic:="Spectra")
  Application.DDEExecute channelNumber,_
  ↙"[Import ""c:\omnic\spectra\absorb.spa""]"
  Application.DDEExecute channelNumber,_
  ↙"[PeakHeight 1600 Shift]"
  returnValue = Application.DDERequest(channelNumber,_
  ↙"Result Array")
  Application.DDETerminate channelNumber
  Worksheets("Sheet1").Range("A1").Value = returnValue
End Sub
```

Note There is a set of Atlμs commands that you use to create Pro macros for the Atlμs application. For information about these commands, including a complete list with descriptions, please see the Macros\Pro on-line help. ▲

Index

h

i

k

l

m

o

p

r

s

t

v